

Everything you wanted to know about REXX but were afraid to ask...

REXX

R E P O R T

Summer 1995

A SPECIAL REPORT FROM THE
PUBLISHERS OF

os/2
Developer
THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

The Power of
Object REXX

**Code Tuning
Techniques**

**Visual REXX Tools—
A Comparison**

**Combining REXX
and PostScript**

BUYERS GUIDE:
**REXX Products
for OS/2**

**REXX
STANDS**

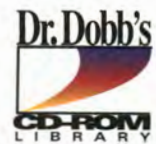
TALL



U.S. \$7.95
Canada \$8.95

A Miller Freeman Publication

Alternative Programming Languages CD-ROM



This CD-ROM is your definitive resource for the latest in cutting-edge programming environments. Gain access to the most innovative, solution-providing languages and tools available today!



Examine the next generation of languages

You need this tool to get up to speed on distributed computing, embeddable languages, increased productivity, and languages of the future.

You get all these languages, compilers, interpreters - Perl, Glish, Smalltalk, Sather, Tcl, M0, Modula-3, ReXX, Lout, Ghostscript, Dylan, Duel, Bob, Neudl, Python, Oberon, Parasol, S-Lang, Quincy, uSystem, Distributed C, the GNU compiler suite—and more!

Put this CD-ROM to work immediately on your development projects:

- **Object-Oriented Development.** Tired of C++? Look no further than Sather, Dylan or Bob and get re-energized.
- **Distributed Computing.** When building distributed systems, languages like Parasol and M0 make your life easier.
- **Multiple Platforms.** Check out Perl and Python, plus a wide range of compilers, including the GNU compiler suite.
- **Text Processing and Document Formatting.** Lout and Ghostscript let you edit and produce complex documents with ease.
- **Prototyping.** When prototyping connected distributed systems, Glish makes it all happen for you.
- **Neural Networks.** Neudl takes the pain out of building complex neural networks.
- **And much more!**

FREE CALL!
800-407-2587
24 HOURS A DAY
7 DAYS A WEEK

All these languages are robust, imaginatively designed, break new technical ground, and come with source code for compilers or interpreters. And you also get the full descriptions and specifications.

SPECIAL BONUS!

With your purchase of this CD-ROM, you'll also receive a copy of the Dr. Dobb's Sourcebook, *Alternative Programming Languages* A \$4.95 value — ABSOLUTELY FREE!

YES! Send me the Dr. Dobb's Alternative Programming Languages CD-ROM immediately. I will also receive the Dr. Dobb's Sourcebook *Alternative Programming Languages* as a bonus.

My price is \$49.95 (plus shipping: \$2.00 U.S./Canada; all other countries \$15.00). California residents, please add \$4.25 sales tax. **Clip out or photocopy this form.**

Name _____

Address _____

City/State/Zip _____

Country _____

Here's how I'm paying (circle one): VISA - MC - AMEX - Check enclosed

Credit card number (include all digits) _____

Expiration Date _____

Signature required _____

RR1

For more
information,
send E-Mail to:

Alternative Programming
Languages CD
info_altcd@mfi.com

FREE CALL: 800-407-2587 24 HOURS A DAY, 7 DAYS A WEEK **FAX ORDERS: 510-372-8582**
E-MAIL ORDERS: order_CDROM@MFI.COM **MAIL ORDERS:** ALT CD, P.O. Box 1525, Martinez, CA 94553
INTERNATIONAL: USE MAIL, FAX 510-372-8582, E-MAIL order_CDROM@MFI.COM, OR CALL 415-655-4252

You can't get more graphic than this.

Introducing the Essential Books on Graphics Programming CD-ROM from the Dr. Dobb's CD-ROM Library

Need to get up to speed on graphics programming—fast? There's only one definitive source you can turn to—*Dr. Dobb's Essential Books on Graphics Programming CD-ROM*!

Get seven of the essential books in graphics programming, with full text, diagrams, graphics, and source code—all on one CD-ROM. From fundamental algorithms to the most complex techniques, this CD-ROM lets you find all the critical information you need for your graphics programming projects.

Texture mapping, color modeling, and morphing, all optimized for speedy 2-D and 3-D graphics programming. Practical image acquisition and processing. High-quality 3-D photorealistic graphics. Digital halftoning. Image synthesis and special effects.

And much more!

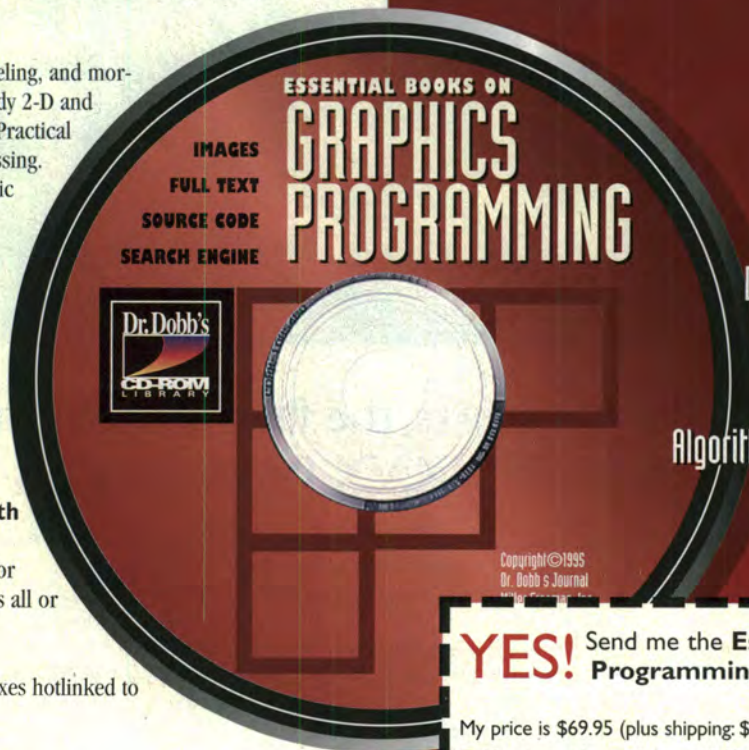
Plus, all the source code on the disks supplied with the books are included.

Your CD-ROM comes with must-have features:

- a powerful search engine for locating information across all or selected books
- tables of contents and indexes hotlinked to their topics
- copy, bookmark, and notes functions

And you get a full money-back guarantee!

**Windows
Platform**



THE DDJ SELECTION: WE DID THE LEGWORK FOR YOU!

Zen of Graphics Programming

Michael Abrash



Photorealism and Ray Tracing in C

Watkins, Coy and Finlay



Practical Image Processing in C

Craig A. Lindley



Applied Concepts in Microcomputer Graphics

Bruce Artwick



Digital Halftoning

Robert Ulichney



Digital Image Warping

George Wolberg



Algorithms for Graphics and Image Processing

Theo Pavlidis



Copyright © 1995
Dr. Dobb's Journal

YES! Send me the **Essential Books on Graphics Programming CD-ROM** immediately.

My price is \$69.95 (plus shipping: \$2.00 U.S./Canada; all other countries \$15.00). California residents, please add \$5.95 sales tax. **Clip out or photocopy this form.**

Name _____

Address _____

City/State/Zip _____

Country _____

Here's how I'm paying (circle one): VISA - MC - AMEX - Check enclosed

Credit card number (include all digits) _____

Expiration Date _____

Signature required _____

RRG I

**24 HOURS A DAY
FREE
800-587-8313
CALL
1 DAY
E-MAIL**

ORDER YOUR CD-ROM TODAY!

ONLY \$69.95 PLUS SHIPPING

FAX ORDERS: 510-372-8582 USE THIS ORDER FORM.

E-MAIL ORDERS: order_CDROM@MFI.COM

MAIL ORDERS: Dr. Dobb's/CD, P.O. Box 1525, Martinez, CA 94553-9802

INTERNATIONAL: USE MAIL, FAX, E-MAIL, OR CALL 415-655-4252

In the Race for Solutions, Finish First



Introducing VisPro/Reports, the first programmable report writer for VisPro/REXX, VX REXX and OS/2 REXX

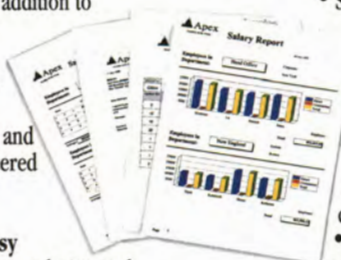
On April 28, 1993 HockWare Inc. shipped VisPro/REXX, the first OS/2 visual programming tool for REXX. Since then, we've achieved many other firsts. VisPro/REXX, VisPro/C and VisPro/C++ are the first REXX, C and C++ tools to incorporate drag and drop programming, an entity-relationship database designer, and a rich set of add-on objects including: spreadsheet, business graphics, formatted entry field, clock and calendar. Whether you're using REXX, C-Presentation Manager, or IBM's User-Interface Class Library, there is a VisPro product to meet your needs.

Yet Another First

VisPro/Reports is the latest addition to the VisPro family of OS/2 programming products. VisPro users will recognize the easy-to-use CUA'91 user-interface and the drag and drop style HockWare pioneered years ago.

Designing Reports is Easy

- WYSIWYG, free-format report design tool.



Report Printing is Easy

- Banded report options allow you to choose from a rich set of headers, footers and groups.
- Label reports allow you to select from a large list of Avery labels.
- Support for multiple line text, derived fields, business graphics, bitmap images, rectangles, lines and ellipses.
- Many object properties to choose from, including any OS/2 font, 16 million colors, line styles, fill patterns and drop shadows.
- Wide selection of field options, International currency and formulas.
- Numerous sample reports included.
- Seamlessly print or preview reports from any OS/2 REXX environment including VisPro/REXX, VX REXX and plain OS/2 REXX.
- Use your favorite REXX-enabled databases and third party libraries for access to DB2, .DBF files, EHLLAPI mainframe connections and more.
- A runtime DLL of less than 100K can be distributed royalty free.

The Complete VisPro Family

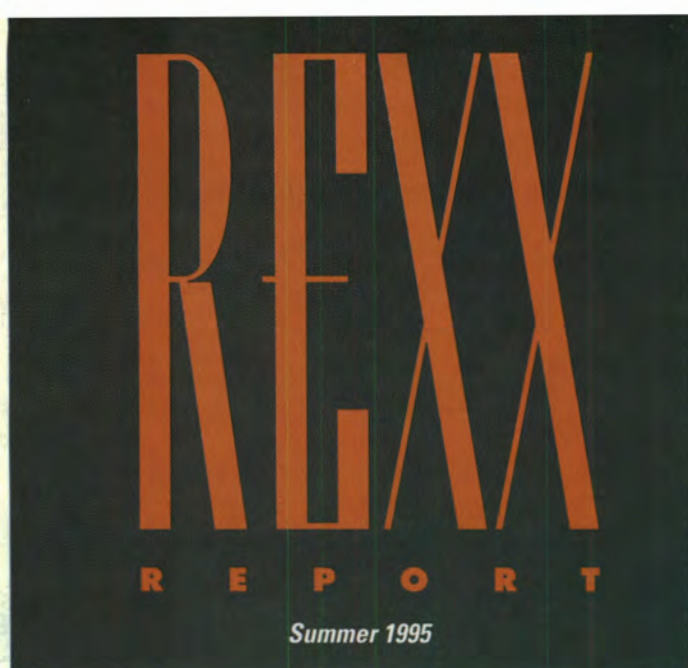
VisPro/Reports	\$199
VisPro/REXX Gold	\$299
VisPro/REXX Bronze	\$99 \$ 59
VisPro/REXX Data Entry Object Pack ..	\$149 \$ 89
VisPro/C or VisPro/C++ SPECIAL!!!	\$399 \$199
VisPro Development Suite	\$599 \$499
(includes VisPro/REXX Gold, C and C++)	

Whether you've already discovered the VisPro family of visual programming products or you're just getting to know us, meet our latest member, VisPro/Reports. And be the first to get where you're going.

Special Offer
VisPro/C \$199
VisPro/C++ \$199
Save over 50%
Offer Expires 10/15/95

HockWare
Putting You in Control

HockWare Incorporated
P.O. Box 336
Cary, NC 27512-0336
919-380-0616
919-380-0757 FAX
Go HockWare on CompuServe
hockware@vnet.net on Internet
<http://www.nando.net/ads/hockware>

EDITOR*Dick Conklin***MANAGING EDITOR***Theresa Mori***EDITORIAL ASSISTANT***Deborah Sommers***EDITORIAL ARTIST***Peter Altenberg***GROUP DIRECTOR***Regina Starr Ridley***PUBLISHER***Peter Westerman***ADVERTISING SALES***Yvonne Labat**(415) 905-2353**Stephen Nikkola**(415) 905-2256**Kristin Morgan**(212) 626-2498***MARKETING MANAGER***Susan McDonald***ADVERTISING PRODUCTION****COORDINATOR***Glenn Sadin***VICE PRESIDENT, PRODUCTION***Andy Mickus***VICE PRESIDENT, CIRCULATION***Jerry Okabe***CIRCULATION DIRECTOR***John Rockwell***CIRCULATION MANAGER***Stephanie Blake***SUBSCRIPTION INQUIRIES***U.S.: (800) WANT-OS2**Foreign: (708) 647-5960*

A SPECIAL REPORT FROM THE
PUBLISHERS OF

OS/2
Developer
THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

**GUI***REXX Trek—A Comparison of REXX Visual Builders 4***COMMUNICATIONS***E-Mail Enabling REXX 9**PDS Transfers Using EHLLAPI in REXX 22***OBJECTS***Intro to Workplace Shell Programming Using REXX 13**Object REXX—HowCool 31**The Workplace Shell: Objects to the Core 42***PRINTING***Combining REXX and PostScript 17***PERFORMANCE***Six Simple Code-Tuning Techniques 35***INTERNATIONAL***Designing REXX Programs for Multiple Languages 39***REXX***REXX Buyers Guide 52**OS/2 Developer: Vol. 7, Summer 1995*

Subscription information: U.S.: One year (six issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-0537). IBM employees and branch office customers can subscribe to *OS/2 Developer* through IBM Mechanicsburg's Systems Library Services (SLSS) using *OS/2 Developer's* order number: G362-0001. POSTMASTER: Please send address changes to *OS/2 Developer*, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. *OS/2 Developer* (ISSN 1073-0729) is published bimonthly by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200. Application to mail at second-class postage rates is pending at San Francisco, CA and additional mailing offices.

© Copyright 1994 by Miller Freeman Inc. PRINTED IN THE U.S.A.

Contact

Dick Conklin
(Editorial)

Miller Freeman
(Circulation)

Peter Westerman
(Publisher)

Theresa Mori
(Managing Editor)

CompuServe

76711,1005, or
OS2DF2 Forum

71572,341

76307,1054

Internet

os2mag@vnet.ibm.com

71572.341@compuserve.com

pwesterman@mfi.com

tmori@mfi.com

un Miller Freeman
A United News & Media publication



Which REXX tool is best for you? It depends. A Trekkie reviews VX-REXX, VisPro REXX and Gpf REXX.

By BLAKE WATSON

REXX Trek— A Comparison of REXX Visual Builders

I have used the various visual REXX tools almost from the beginning, and I sometimes get asked which of the three tools is the best: Watcom's VX-REXX, Hockware's VisPro REXX, or Gpf REXX. The answer is usually a firm: "It depends."

Not only does each of the tools have its strengths and weaknesses, each has sufficiently different interfaces, whereby one tool might drive you nuts while another tool seems intuitive to you, even though both offer comparable features.

To illuminate some of these differences, I pulled out the rusty old Star Trek game, which is a standard program I write to test out various environments. For those of you who don't know, the Star Trek game, developed in the late 1960s at Carnegie-Mellon University in Pittsburgh, Pa., simulates a military conflict in space where the player attempts to destroy the alien invaders within a certain amount of time. The player has a limited amount of energy to deploy weapons and shields and move through space. The player can refuel at outposts, which may be attacked and destroyed by the aliens. I can write this game in my sleep, which leaves me free to concentrate on the development environments and, at the same time, lets me test how the environment handles subroutines, objects, and other interesting items.

The notes and comments expressed herein are based on the following versions of products: Gpf REXX 1.0 (level 04), Hockware VisPro/REXX Gold 2.03, and VX-REXX Client/Server Edition Version 2.1. Tests were done under OS/2 2.1, 2.11, and OS/2 Warp for Windows.

BUILDING THE GUI

Let's start the comparison by looking at the basic GUI for the game. I typically design it as a three-window game, with a Command

Panel (CP), from which the user issues commands, and two other windows, one representing Long Range Scanners (LRS) and the other representing Short Range Scanners (SRS).

Space in this game is an eight-by-eight grid, where each division (called a quadrant) is divided into an eight-by-eight grid of sectors. So, space consists of 64 quadrants, and each quadrant consists of 64 sectors.

Figures 1, 2, and 3 show how the interface looks under Gpf REXX, VisPro/REXX, and VX-REXX, respectively. The CP is usually just a window full of buttons and text controls, and the two sensor windows work nicely as value sets.

As you can see, I've modified the game slightly from its traditional format to make it easier to use. The commands are actually issued from the LRS and SRS windows: The user clicks on the value set to the right to indicate a command (on the SRS, set destination, go, target phasers, and target photons; on the LRS, set destination and go) and then double-clicks on a spot on the value set to indicate the desired location. The command panel is only used to offer information and set the phaser and shield strength. Now we'll look at the issues that arose while building each of the interfaces under the various products.

GPF REXX

The interface Gpf REXX uses is very OS/2 1.3 oriented, and yet the polish of time shows up when you use it. Yes, it's heavily menu and dialog driven, but while this makes it somewhat less aesthetic, perhaps, it doesn't typically make it harder to use. In some ways, in fact, it seems easier to use—perhaps only because it is based on Gpf's C GUI builder, which is one of the oldest around for OS/2.

One place where it is definitely harder to use than the other two is in the setting up of a value set. Gpf requires a name for each cell in the value set or you cannot access that cell at run time. To add a component to a Gpf value set, you start from a dialog showing a listbox of all components for that value set, then you click on an Add button to bring up a dialog for that component, and then you click OK to dismiss the dialog.

Naming 64 cells (twice) was the least-pleasant aspect of working with Gpf. In fact, I didn't do it. I just accepted the default names that Gpf supplied and wrote the code to work around those names (which are fortunately serial).

Actually, the least-pleasant aspect of working with Gpf is reserved for those using 542x Cirrus cards under OS/2 2.1 or 2.11. Gpf and the old Cirrus drivers do not cohabit well, and that will be extremely unpleasant. Under Warp, those problems vanish, however.

One of the more pleasant aspects of working with Gpf REXX was its recognition and handling of presentation parameters. Gpf stands out from the other products in its ability to create and work with presentation

parameters. (You can assign a group of controls the same presentation parameters and give them a common look and feel far more easily than you can setting each control's individual color and font fields.)

Even under Gpf though, you couldn't change a group of controls all at once so that they all shared the same presentation parameters. (Unfortunately, none of the tools would let you group change the attributes of an object—except for size and position—which makes creating a consistent nondefault look between the controls a real pain in VX-REXX and VisPro/REXX.) So even when programming with Gpf, the first thing you would do is define what controls are going to look alike and assign them a presentation parameter when you created them. Then later, you would go back and modify the presentation parameters to customize the look of your application.

Of the three products, Gpf seems to acknowledge the underlying API the most. For example, you can place a label control on your form, then with a click of a dialog setting change it to any of the valid forms of the WC_STATIC window class. In other words, you can turn your label into a

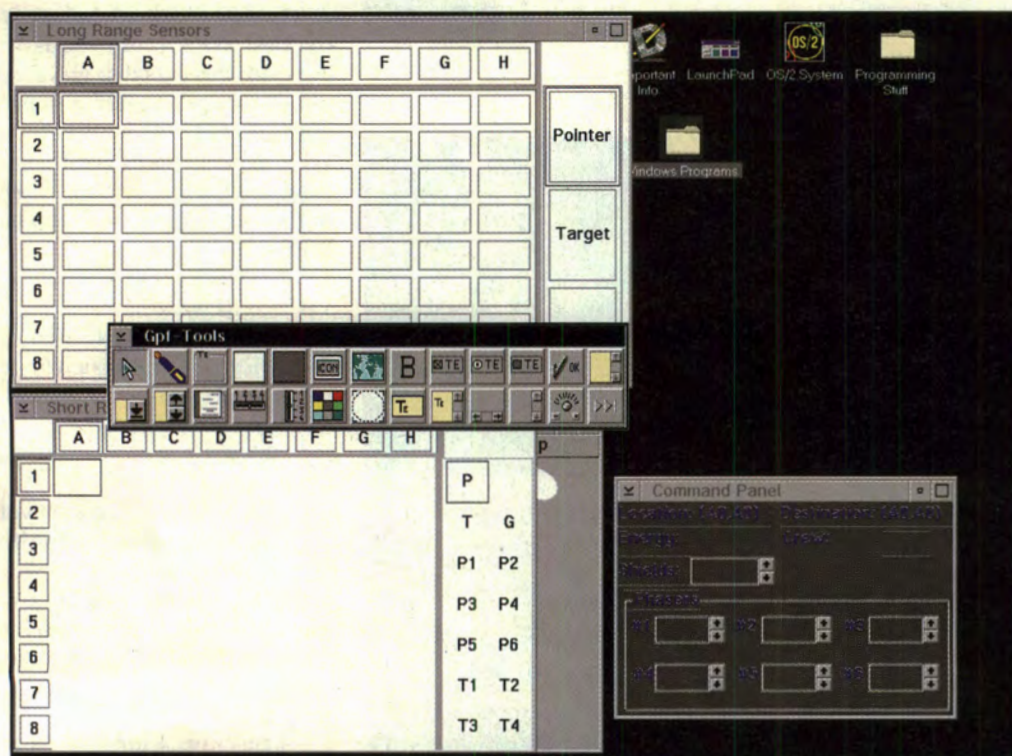


Figure 1. How the interface looks under Gpf REXX.

bitmap, a rectangle, or whatever. This can give you a very strong feeling of control, especially if you are comfortable with the PM API. Of the three products, I can also only nervously recommend Gpf, for while it is a good product, it is still at 1.0 after well over a year in release.

VX-REXX

Setting up the program in VX-REXX was a breeze, for the most part. Unlike the dialog/menu-oriented Gpf REXX, VX-REXX has lots of windows, which can actually make it difficult to find your way around when developing an

application with three windows of its own. I often found myself moving secondary windows around to get to the environment's main window.

The VX-REXX environment strikes me as somewhat Windows-y. A better description might be Visual Basic-y. I've heard Visual Basic programmers can sit right down with VX-REXX and feel completely at home. This can be a good point and a bad point, depending on how much of an OS/2 snob you are. I've never used Visual Basic, so I was initially annoyed by what seemed to be a nonOS/2 way of doing things. But VX-REXX is

hardly difficult to get used to, despite this.

Of the three products, VX-REXX has two oddities that stand out: First, there is no default sizing option in the menu. Watcom representatives have told me that VX-REXX is entirely user-configurable, and you can actually add a sizing option to the menu using a macro you can download from their forum on Compuserve.

Second, the copy option requires you to use the clipboard. Both VisPro and Gpf have drag-and-drop copy methods, but in VX-REXX you must select an object, copy it to the clipboard, and paste it back in.

Also, I found what appeared to me to be a bug in the VX-REXX environment. When I gave my value set too many items (by accident), the control vanished. VX-REXX gave it a size and location of zero. In the process of trying to resize it, I accidentally closed the settings notebook, and the control was lost to me forever. To save my work and not be stuck with an invisible orphan control, I had to copy the other controls to the clipboard, delete the old window, create a new window, and copy the controls back in.

Fortunately, it wasn't a complex form. Nonetheless, this incident was disturbing.

VISPRO/REXX

VisPro/REXX is the best WPS-oriented designer of the three. In fact, it's one of the best WPS applications I've seen. But even this cuts two ways. For example, you can't resize a control with both mouse buttons like you can in WPS, which is a minor thing, but jarring precisely because you're expecting everything to work exactly like WPS does.

And talk about multiple windows! VisPro/REXX does everything with folders and views, so you can have three views open on one window of an application, and you'll always have at least one additional folder open, and probably two or more, to contain your project files.

VisPro/REXX requires you to use your WPS management skills to the fullest. When you're done with a window, for example, you close it— even if you might come right back to it in a few minutes. They don't take long to open, and this practice helps you keep your desktop clean.

I ran into a kind of "that's good but..." phenomenon a lot while working with VisPro/REXX. For example,

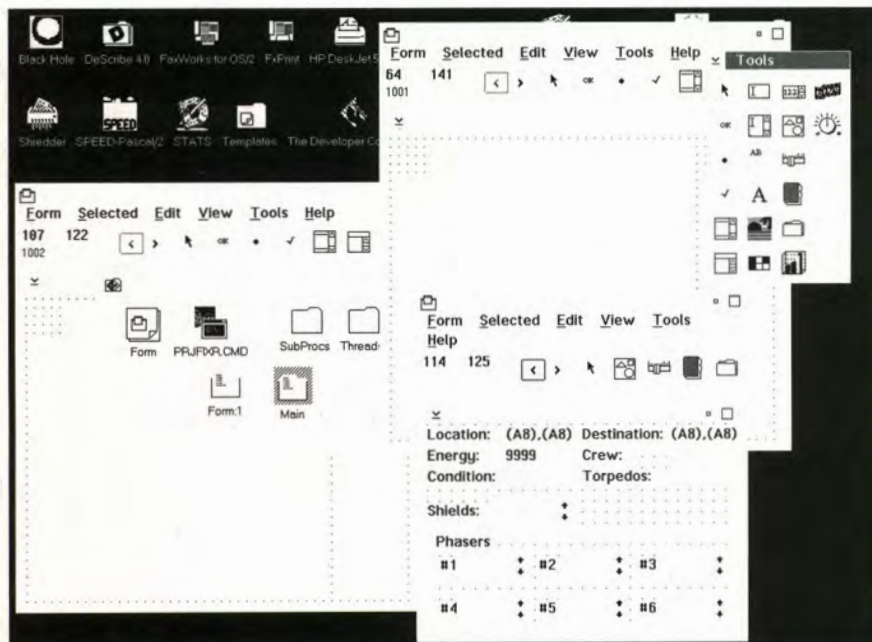


Figure 2. How the interface looks under VisPro/REXX.

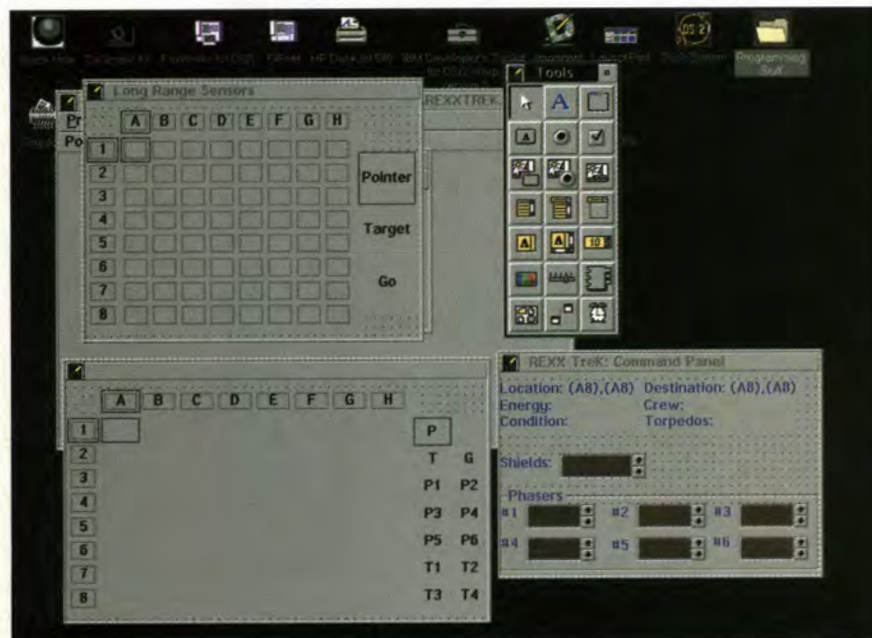


Figure 3. How the interface looks under VX-REXX.



options for a placed component are set by opening up a settings notebook. Very WPS-like, which is good.

At the same time, setting something in the settings notebook did not necessarily mean that it would actually change. The settings notebooks were generally not as reliable as doing something programmatically at run time.

Also, sometimes, in changing the settings of many objects at once, it was easy to regret calling up several dozen settings notebooks at once. In fact, if there were too many, they would all collapse back down.

I found another good example of good news/bad news in the value set control. VisPro/REXX was the only environment that allows accessing a value set by row and column, which was great. At the same time, there was no way to initialize the value in the value set, and VisPro/REXX doesn't support bitmaps at design time.

Of the three, VisPro/REXX is the most ragged, in that it is most likely to do something unexpected or undesired. The controls don't always snap to the grid, giving some created windows an uneven look. Also, given two items at the same location, VisPro/REXX takes the one underneath when you click on the location.

CODING THE APPLICATION

The biggest disappointments from all three tools came when I went to integrate actual REXX code into the GUIs. Two layers of code are really involved: the first is actual domain code, used to implement the game rules, and the second is code used to interface the domain code with the GUI.

Of course, the latter is completely different from product to product, as you might reasonably expect. The tools were not really built with interproduct compatibility as a consideration.

But the tools were not built with the idea that you would have a set of REXX functions in one or more text files that could be maintained separately from the GUI code. Gpf REXX and VX-REXX let you stash your subroutines in "precompiled" function libraries, which is handy, but neither of these environments will (for example) create one of these libraries from your text files and let you pick from routines graphically. It's all done the old-fashioned way.

My main hope at this point is that OO REXX will aid in this area and that it will be possible to set up class definitions that all three can call and,

thereby, minimize compatibility problems. In fact, I imagine OO REXX could define a "standard GUI class library" that all three could conform to, perhaps letting users port all their code painlessly from product to product. Now I'll discuss some specific situations I encountered implementing the program with the three products.

GPF REXX

Gpf REXX has my favorite approach to handling basic component manipulation calls. You often don't have to do it at all. Instead, you get an "action" tree that lets you plug in certain actions to certain events. To get the SRS and LRS windows to open with the CP window, I just clicked on a tree branch to indicate the event and then selected from a list box of possible actions.

The list box includes your own user function "objects," as they call them, which means you'll never miss a function name when interacting with Gpf REXX. Unfortunately, the "User Function" item of the list box—the one you'll select the most—is always at the end, so you always have to scroll down to the bottom to select it.

Domain subroutines in Gpf are integrated with the program file—you have to cut and paste your existing routines into your new Gpf REXX programs. The program file itself is in a proprietary format. For some people, this is a big drawback. For me, since I have to paste in my own routines anyway and I have no use for the Gpf calls, I don't much care.

Gpf documentation is geared toward experienced programmers. Some of it seems to be lifted from the C version of the product, even. The documentation contains no object reference, just a function reference, so you'll probably end up doing some digging to find out how to do a particular thing for a particular object. (It took me a long time to work out setting value set items, for example.)

Gpf REXX has the worst development cycle of the three products. If you select save your file, Gpf will ask for verification. By default, if you try to close Gpf, Gpf will ask you if you really want to exit. (It sticks this same over-prompting into your application, too.)

Far worse than that, though, is Gpf's tendency to close windows. If you save your project, your windows close. If you run your project, not only do your windows close, your project closes and you must reopen it to continue work. None of the three products

offers the courtesy of a "recall" menu item, either.

VX-REXX

VX-REXX has, far and away, the best documentation of the three products. In all the time I have used it, I've found only a few vagaries in the documentation, one of them (ironically enough) with the SetAttributes method that I had to use to set the items in the value set.

But while it was difficult sometimes to figure out how to perform some specific task in the other two products, that was rarely the case in VX-REXX (the above-mentioned SetAttributes method being the only case in this program).

And just about anything is possible under VX-REXX, too. You can create a new control at run time, something not possible at all under Gpf REXX and something I could never figure out whether it really was possible under VisPro/REXX.

You can make your labels respond to double-clicks under VisPro/REXX, if that's what you want to do. The other products restrict you to the "normal" events for objects. (It is possible to respond to *any* PM events from a Gpf app, but not easy to deviate from the normal.)

At the same time, I actually uncovered a bug in the value set control while writing this program, which I hope has been corrected. (Setting a value set item to a null bitmap should have drawn the background color over anything that was already there, but that didn't happen in the right most column and bottom row!)

As with Gpf REXX, VX-REXX requires you to cut and paste your routines into a section editor. VX-REXX doesn't give you a really good way to plug these calls in, either. In some ways, VX-REXX's code integration is the weakest of the three.

VISPRO/REXX

The documentation for VisPro/REXX improved greatly between 1.0 and 2.0. I didn't really have much trouble figuring out how to do things using the new manuals.

VisPro/REXX comes with a kind of automatic organization. Every project has a sub-proc folder where subroutines are each kept as discrete files. The advantage to this approach over the other two is obvious: you can just drag your existing subroutine code into the sub-procs folder for any VPR application. Unfortunately, VPR

doesn't recognize HPFS long filenames, so your subroutines are restricted to eight characters.

In addition, I found myself doing a lot of messaging back and forth between the VisPro windows. In the Star Trek game, the LRS gets updated after certain events, and either of the other windows may want to call the LRSUpdate function.

So, my initial thought is always: make it a sub-proc. The latest VisPro/REXX, happily, has support for dragging-and-dropping code when creating sub-procs. Older versions required you to manually insert code in sub-procs, which made them a less-attractive feature.

The other, better option is to have the windows send messages to each other. I have the SRS and CP windows message the LRS when its time to update. But the support for messaging ends with "you can do it." VisPro/REXX doesn't give you any way of organizing your messages—which you will surely want for a project of any size.

DECISIONS, DECISIONS

It's very difficult for me to proclaim

one product better than another, because they are generally compatible products that excel or lack in certain areas, and all have peripheral elements that make them more attractive (such as VisPro REXX's database diagramming tool).

None of the products offers a particularly good way to integrate existing REXX code into the GUIs they build. This is unfortunate and has always been a barrier to me personally developing really large programs using REXX.

But what is likely to make or break a certain product for any individual is the user interface. And interfaces are like programming languages or operating systems—it's difficult to predict what any person will like.

I can't offer much in the way of metrics: For example, I could build the interface for the game in Gpf REXX in about half the time of the other tools, with VisPro REXX taking the longest—the opposite of what I would have guessed. You can't tell whether this reflects some personal bias or is a reflection of the smoothness of Gpf REXX vs. the raggedness of VisPro REXX—a raggedness that

seems to have mostly been cured between the 2.03 version I wrote the program with and the 2.11 that is currently available, while Gpf REXX still languishes at 1.0.

Actually coding a program in the three was basically the same in terms of time and effort. They each pleasantly surprised me from time to time, and they each drove me up a tree quite often. If nothing else, this highlights the dubiousness of proclaiming "a winner" among the three.

When the products first came out, a lot of leapfrogging and competitiveness seemed to have died down in recent months. All three companies have products other than their REXX GUI builders, and I suspect they have been focusing on those of late.

However, I expect to see a surge of new developments made to all three tools in the few months following the official availability of OO REXX. I hope so, because REXX still has a lot of untapped potential.

Blake Watson is the author *OS/2 Warp Programming for Dummies* and an upcoming book on *IBM Smalltalk*. You can reach Blake electronically at 70303,373@CompuServe.com.

AUTOMATE!

Windows 95
Windows NT
Windows 3.1

Info REXX

REXX Information at your fingertips!

Multimedia help file chock-full of REXX programming examples, language references for Personal REXX, OS/2 REXX, Object REXX, REXX API functions. Includes listings for over 140 REXX products, 80 REXX vendors, magazine articles, REXX on the Internet and much more!

\$20
OS/2
Windows
Windows NT

with ENTERPRISE REXX

REXX programming on any Windows platform Is EASY!

A fast, complete REXX interpreter
Intel 80x86
DEC Alpha
MIPS RISC

\$99
32-bit and 16-bit

INTER REXX newsletter

learn how to code EHLLAPI, VX-REXX, Object REXX, ISPF dialogs and more.
Source code for each article available online!

\$24
12 Issues

REXXANNE

A truly amazing REXX Compiler for DOS, OS/2, Windows, Windows NT & Windows 95!

Order today from the REXX store!

Call 800-741-4322

Outside USA call 1(303) 442-7700

Pelt Industries
Software Experts
8027 N 41st Street, Longmont, CO 80503
Fax: (303) 442-3198 REXX-R-U BBS: (303) 440-1351

REFERENCES

VisPro REXX, Hockware, P.O. Box 336, Cary, N.C. 27512-0336 E-mail: 71333,3226@CompuServe.com

VX-REXX, Watcom, North America 415 Phillip St., Waterloo, Ontario CANADA, N2L 3X2. Sales: 800/265-4555, Info: 519/886-3700, Fax: 519/747-4971, Compuserve: GO WATCOM

Gpf REXX, Gpf Systems Inc., 300 Falls Rd., Moodus, CT 06469, 203/873-3300, Fax: 203/873-3302.



REXX is ideal for e-mail applications. Vendor Independent Messaging (VIM) makes it possible. Here's an introduction to writing REXXVim applications.

By MARK RAMSEY

E-Mail Enabling REXX

One of the most powerful features of REXX is support for external functions. Many application packages now support REXX through the use of REXX extension libraries. IBM LAN Server 4.0, for example, provides access to the full interface using REXX external functions. REXX libraries are available to support everything from the dbase file system to a full communications library. Because of its extensibility, REXX is an ideal development language for e-mail enabled applications.

The Vendor Independent Messaging (VIM) specification, sponsored by Apple, Borland, IBM, Lotus, MCI, Novell, Oracle, and WordPerfect, defines a nonproprietary, multiplatform solution for providing messaging services within applications. VIM allows a single interface to be used across several e-mail products, such as IBM UltiMail, Lotus Notes, Lotus cc:Mail, or Netware MHS. VIM support is available on many operating system platforms, such as DOS, Windows, OS/2, Macintosh, Netware NLM, and UNIX.

Lotus offers the VIM Developer's Toolkit to support VIM application development using C/C++ or Visual Basic. Innovative Business Technologies offers REXXVIM to support VIM development using OS/2 REXX. The rest of our focus will be in the REXX environment; however, the APIs are the same in all of the environments.

DEFINING THE E-MAIL APPLICATION

Our application will be a system monitoring agent. The agent will run on remote OS/2 systems to prevent "out of disk space" problems. The users on the remote systems all access our company e-mail system, which will be the pathway for our agent to send warning messages.

In our example scenario, it is important

to note that all of the remote systems do not need to be using the same e-mail package. As long as all of the connections are in place for each of the mail systems to exchange mail, the individual packages are not important. Some of the systems may be using Lotus Notes within a departmental environment, while others may be using cc:Mail as the mail service.

For our discussion, all of the remote systems must use an e-mail system that supports VIM. It is possible to use VIM with systems that support the Microsoft MAPI mail API, but we will leave that to a future article.

Because of the flexibility of sending warning messages through the e-mail infrastructure, the remote systems do not need to be connected to a common network. This allows our application to send warning messages from systems that are on many different networks, as long as those systems can send e-mail to the main system. Because of e-mail gateways and bridges, our application will be able to monitor remote systems that connect to a service provider to route messages to the main company location.

Our example monitoring agent will be limited to tracking remaining disk space. If the remaining disk space on a remote system falls below 5%, the agent will send a warning message to the Administrator ID. The basic monitor application can easily be expanded to track other components on the remote system or even problems on remote servers.

BUILDING OUR APPLICATION

The first step in our application is loading the external function library for VIM support. The code fragment shown in Figure 1 loads all of the functions in the REXXVIM library. The RxFuncAdd statement is used to load the RxVIMLoadFuncs external function. Calling the function loads the rest of the functions in


```

/* Load the REXXVIM extensions to REXX */
rc = RxFuncAdd('RxVIMLoadFuncs', 'REXXVIM', 'RxVIMLoadFuncs')
if rc > 0 then do
  Say
  Say ' REXXVIM failed to load, return code' rc
  Say
  signal done
end /* Do */
call RxVIMLoadFuncs

```

Figure 1. Loading the VIM extensions.

```

/* Variables */
low_percent = 5
temp_file = 'DIRLIST.txt'

.
.
. (later in program)
.

parse value SysDriveInfo('C:') with drive free total label

free_percent = FORMAT(free/total * 100,2,0)

if free_percent <= low_percent then do

  /* Text to send with message */
  TextMsg = "Drive Size:" total " Free:" free " Label:" label

  /* Create file listing for root directory */
  'dir c:\ >' temp_file

  /* Call the procedure to send message to administrator */
  call SEND_MESSAGE

  /* Remove the temp directory file */
  'erase' temp_file

end

```

Figure 2. Determining available disk space using SysDriveInfo.

```

/* Open a session with the postoffice using the parameters */
/* specified. */
/*
/* db_path - path to the message container database */
/* user_name - name of the e-mail id to open session */
/* user_pw - password for the user_name account */
/* Session - will contain the session pointer after the */
/* completion of the function call */

rc = RxVIMOpenSession(db_path,user_name,user_pw,'Session')
if (rc > 0) then signal VIMError

```

Figure 3. Opening and logging into a session using the VIM interface.

the library. If the RxFuncAdd statement fails, our application will go immediately to the cleanup code.

For our application, we could also choose to load only the nine functions that are needed. These functions

would be loaded through nine individual RxFuncAdd calls. With this approach, the cleanup phase would also require nine calls to the RxFuncDrop function to release the individual VIM functions.

MONITORING DISK SPACE

In our application, the monitoring agent will send a warning message when available disk space falls below 5%. Using the SysDriveInfo function provided with the REXXUtil library, we are able to obtain drive size and available disk space. The code fragment shown in Figure 2 calculates the percentage of available disk space. Our application can use this figure to determine if it is time to send a warning message.

If the available percentage reaches 5% or less, our application must send a warning message. The code fragment shows the application creating the message text to send to the administrator and creating a file listing of the root directory. Let's look at the steps involved in sending the message once the threshold has been met.

Initializing VIM. Once all of the necessary functions are loaded into the REXX environment, we must initialize the VIM subsystem. The following must be the first VIM call made:

```
rc = RxVIMInitialize()
```

With the initialize, like all of the VIM calls, a zero return code indicates success. The RxVIMInitialize function will return a nonzero return code only if a problem exists with the underlying mail system.

Opening A Session. The next step in our application is establishing a session with the actual e-mail system. The RxVIMOpenSession function opens and logs into a session using the VIM interface (Figure 3).

The call is comprised of three parts: database path, user name, and password. The database path is the location of the e-mail file containing message information. In Lotus cc:Mail, this would be the location of the MLAN-DATA and CLANDATA files. In Lotus Notes, this would be the location of the mail database file. The name and password parameters must be for a valid user on the e-mail system.

For our application, each remote system will use the name of the user on that system. VIM supports multiple connections using the same user name, so our program will not interfere with normal e-mail functions. The remote users can continue to use whatever e-

mail access product is currently in place, while our application runs in the background.

The RxVIMOpenSession function returns the value of the session in the REXX variable Session. This variable contains a pointer to the session opened by the function. We will use this variable in future VIM calls. The session variable allows our applications to have multiple sessions opened with VIM from different concurrent REXX applications.

Creating the Mail Message. We are now ready to create the actual warning message. First, we must create a new message shell using the RxVIMCreateMessage function. The function requires the session pointer, which was returned from our RxVIMOpenSession call, and the message type to create. The current implementation of VIM supports only mail message types, so the type will always be VIM_MAIL. The function will return a message pointer for us to use with other message functions. We will use the variable Message to contain the value of the message pointer.

Once the message shell is created, we are ready to compose the warning message. Our first action will be to specify the message header information. The RxVIMSetMessageHeader function will be used to set the subject and priority of the message. Figure 4 shows the code fragment to create the message shell, set the subject of our message to "Disk Space at X percent," and set the priority to high. The RxVIMSetMessageHeader function is called once to set each attribute.

All of the VIM functions use keywords to specify and return values. This allows the same application to run on multiple platforms without the need to address precision. The keywords allow programs to ignore the actual values and focus on the function itself. The two keywords used in the RxVIMSetMessageHeader are VIMSEL_SUBJECT and VIMSEL_PRIORITY for subject and priority, respectfully.

Next, we will set the recipient of our message using the RxVIMSetRecipient function. This call allows us to address the message to the Administrator ID and even send a copy to another ID if necessary. For our application, we will send the warning message to an ID called System_Monitor. This call, as you can see in Figure 5, also uses the Message pointer variable from the open session call.

In our application, the System Monitor ID will be defined on all of the

```

/* Create a new message of type VIM_MAIL */
/*
/* Session - session pointer from the open function call */
/* msg type- VIM_MAIL only supported type */
/* Message - will contain the message pointer after the
/* completion of the function call */

rc = RxVIMCreateMessage(Session, 'VIM_MAIL', 'Message')
if (rc > 0) then signal VIMError

/* Set the subject of the message */
/*
/* Message - msg pointer from the create function call */
/* Attribute - Keyword for attribute to set */
/* Value - Value to use */

hdrtext = "Disk Space at" free_percent "percent"
rc = RxVIMSetMessageHeader(Message, 'VIMSEL_SUBJECT', hdrtext)
if (rc > 0) then signal VIMError

/* Set the priority to high */
/*
/* Message - msg pointer from the create function call */
/* Attribute - Keyword for attribute to set */
/* Value - Value to use */

rc = RxVIMSetMessageHeader(Message, 'VIMSEL_PRIORITY',,
                             'VIM_HIGH_PRIORITY')
if (rc > 0) then signal VIMError

```

Figure 4. Creating a shell message and setting the message header information.

```

Setting the message recipient

/* Set the recipient to the administrative user */
/*
/* Message - msg pointer from the create function call */
/* class - keyword for class of recipient (TO,CC,BCC) */
/* type - type of recipient (One or group name) */
/* entity - name type */
/* adrbook - addressbook for name */
/* name - userid to send (in local addressbook) */
/* adrtype - type of addressbook entry (one or group) */
/* adrname - full address entry (Fred Smith at ABC_PO) */

rc = RxVIMSetMessageRecipient(Message, 'VIMSEL_TO', 'VIMSEL_ENTITY',,
                              '', '', admin_name, '', '')
if (rc > 0) then signal VIMError

```

Figure 5. This call uses the message pointer variable from the open session call.

remote post offices. This will allow us to specify the ID only in the RxVIMSetRecipient function. The function provides the flexibility to specify the full name and address of the recipient if it is not included in the local post office.

The last part of our message cre-

ation is adding text to the message to provide the administrator some additional information. In the code fragment shown in Figure 6, we provide the drive size, free space, and drive label. All of these items are returned by the call to SysDriveInfo.

Adding message text and directory listing to the message using RxVIMSetMessageItem

```

/* Add the text to the message */
/*
/* Message - msg pointer from the create message function */
/* Class - class of the item (here it is part of note) */
/* Type - item type (here we are adding text) */
/* Flags - this is native text */
/* Name - name or title of item */
/* Item - text of the item */
/* Filename- not used for this call */

rc = RxVIMSetMessageItem(Message, 'VIMSEL_NOTE_PART', 'VIM_TEXT',,
                          'VIMSEL_NATIVE', 'Text', TextMsg, '')
if (rc > 0) then signal VIMError

/* Add the file attachment to the message */
/*
/* Message - msg pointer from the create message function */
/* Class - class of the item (here it is a file attach) */
/* Type - item type (here we are adding text) */
/* Flags - this is native text */
/* Name - name or title of item */
/* Item - not used for this call */
/* Filename- filename to use for the attachment */

rc = RxVIMSetMessageItem(Message, 'VIMSEL_ATTACH', 'VIM_TEXT',,
                          'VIMSEL_NATIVE', 'Directory', '', temp_file)
if (rc > 0) then signal VIMError

```

Figure 6. The drive size, free space, and drive label is provided.

The second call to RxSetMessageItem attaches the DIRLIST.txt file to our message. The file is a listing of the root directory on the remote system. If we change the directory command to include subdirectories, the file attachment would be a directory of the entire disk drive. The system administrator can use this information to assist the remote user in system cleanup.

Sending the Mail Message. Now that the message is complete, we are ready to send it on its way. The RxVIMSendMessage submits the message to the e-mail system, using the Message pointer:

```
rc = RxVIMSendMessage( Message )
```

Once the message is sent, the e-mail system takes control of routing the message to its destination. All message resources are released, and the message pointer is no longer valid.

Cleaning up. After we have finished sending our message, we must close the active session and end our VIM connection for this process. The RxVIMCloseSession closes the active VIM session and frees all of the VIM session resources. The function uses the session variable to close a specific session.

The RxVIMTerminate function ends the VIM connection for the process. This function is the counterpart to the RxVIMInitialize function. It is part of the overall VIM cleanup process.

Upon exiting our application, we should make a call to RxVIMDropFuncs. This function releases all of the VIM functions from the REXX environment. Failing to make this call will cause the RxFuncAdd call to fail for future executions of our application.

Hey, What's Wrong? Error Processing. If any of the VIM functions complete with a nonzero return code, the VIM specification provides a function to determine the cause of the failure. In our application, a nonzero return code will cause a jump to the VIMError procedure. The VIMError procedure calls the VIMStatusText function to obtain additional information regarding the failure.

SUMMARY

Now that we have a fully functional e-mail enabled REXX application, our system monitor can be expanded to perform many other functions. The application could monitor network errors and forward the error log to a network administrator. A similar application

could forward a status message daily regarding the performance, audits, print jobs, and so on from a remote server. Our example application is just the beginning in the world of REXX e-mail.

Mark Ramsey founded Innovative Business Technologies Inc. (IBT) in 1983. He has been actively involved with OS/2 consulting and development since 1987. Ramsey has a degree in computer science and business administration from Capital University. IBT offers several REXX development tools, including the REXXVIM toolkit described in this article. Mark can be reached on CompuServe at 74563,1530 or at (614) 791-9055 Ext. 11.

REFERENCES

The full system monitor application is posted in the OS2DF2 forum on CompuServe as RAMSEY.ZIP. A demo version of the REXXVIM toolkit is also in the forum under RXVIM.ZIP.

The Vendor Independent Message Specification v.1.0, by Apple, Borland, IBM, Lotus, MCI, Novell, Oracle, and WordPerfect.

REXXVIM: The VIM Toolkit for OS/2 REXX, by Innovative Business Technologies Inc.



Control of OS/2's Workplace Shell is easy, thanks to REXXUtil. Here's object-oriented programming you can do today.

By GOWRI SANKAR SIVAPRASAD

Intro to Workplace Shell Programming Using REXX

The OS/2 Workplace Shell (WPS) provides interfaces to programmers that can be used to manipulate the objects in the Workplace Shell desktop (hereafter referred to simply as the desktop). For REXX programmers, this comes in the form of the REXXUtil DLL.

REXXUtil is a dynamic link library that provides a set of functions to manipulate the WPS objects. This DLL requires OS/2 2.x or above with OS/2 REXX installed. I will discuss how to use the REXXUtil functions in Workplace Shell programming.

To use these functions, first the REXXUtil library must be loaded. The built-in function `RxFuncAdd` can be used for this, as shown here:

```
call RxFuncAdd "SysLoadFuncs", "REXXUtil",
"SysLoadFuncs"
```

This loads the complete REXXUtil library, making all the functions in the library usable by all OS/2 sessions.

If the services of just one function is required, then we can load that function, as in

```
call RxFuncAdd "SysCreateObject" , "REXXUtil",
"SysCreateObject"
```

This just loads the `SysCreateObject` function in the REXXUtil library.

Later, I will introduce the different WPS objects and how to create, destroy, and update different kinds of objects. Toward the end of this article, references to some freeware and shareware utilities for manipulating WPS objects are given.

INTRODUCTION TO WPS OBJECTS

The OS/2 Workplace Shell provides many different types of objects. The most com-

mon ones are the Desktop, Folder, Program, Shadow, and Datafile objects. The Folder object is a placeholder to store other objects. Program files are a shortcut to execute programs. Executables can be associated with Program objects and get executed when users double-click on the Program object's icon. The Shadow object provides a mechanism to duplicate an object at another location, with the same settings. The shadow object behaves in the same way as the original. It gets deleted when the original object is deleted. Any number of shadows can be created for a given object.

Each object in the system belongs to a particular class. Folder objects belong to class `WPFolder`. Each class is given a unique name. A class has to be registered before any objects belonging to that class can be created.

The REXXUtil package contains a `SysQueryClassList` function that retrieves all the registered classes in the system.

Syntax : `call SysQueryClassList stem`
Parameters : stem the name of the stem variable to hold the list of all

```
class 10 is WPObject PMWP
class 11 is WPSysSystem WPCONFIG
class 12 is WPFileSystem PMWP
class 13 is WPDataFile PMWP
class 14 is WPProgramFile PMWP
class 15 is WPFolder PMWP
class 16 is WPDives PMWP
class 17 is WPShreder PMWP
class 18 is WPDisk PMWP
```

Figure 1. Common object classes.


```

classname = "WPFolder"
title = "MyOwnFolder"
location = "<WP_DESKTOP>"
setup-string =
"OBJECTID=<MY_OWN_FOLDER>;ICONPARAM=
myownfolder.ico;"

rc =
SysCreateObject(classname,title,location,
setup-string)
if rc = 1 then
    say " MyOwnFolder : Created"
else
    say " Problem in creating
MyOwnFolder. rc = " rc

```

Figure 2. Create a new folder object.

```

registered classes in the system
Example : call SysQueryClassList
"allclasses."
*/
do i = 1 to allclasses.0
    say "class " i " is " allclasses.i
end

```

Figure 1 shows an example output from this piece of code.

REGISTERING/DEREGISTERING A CLASS

The functions SysRegisterObjectClass and SysDeRegisterObjectClass can be used to register and deregister an object class. Registering a class makes it part of the system (until it is deregistered), and objects of that class can be created

from then on. A new class is implemented as a DLL and registered as shown here

```
rc=SysRegisterObjectClass("MyNewClass",
"MyNewClassDLL")
```

The first parameter is the name of the class, and the second parameter is the name of the module that contains the class implementation. The function returns 1 (true) if the class was registered and 0 (false), otherwise. Similarly,

```
rc=SysDeregisterObjectClass("MyNewClass")
```

deregisters an object class.

CREATING A NEW OBJECT

Instantiation of an object class is called simply "creating a new object." A new object has associated with it the class it belongs to, title, location, and a set of parameter settings defining the behavior of that object.

Here's how the function SysCreateObject can be used to create a new object.

Syntax:rc= SysCreateObject(classname, title, location <,setup-string>)

The parameters of this function are as follows:

- Classname: The name of the (regis-

```

classname = "WPProgram"
title = "MyProgram_in_MyOwnFolder"
location = "<MY_OWN_FOLDER>"

setup-string = "PROGTYPE=FULLSCREEN; " ,
"EXENAME=c:\myown\myprogram.exe; " ,
"PARAMETERS=c:\myown\mydata.dat; " ,
"ICONPARAM=myownprogram.ico;"

rc =
SysCreateObject(classname,title,location,
setup-string)
if rc = 1 then
    say " MyProgram : Created"
else
    say " Problem in creating
MyProgram. rc = " rc

```

Figure 3. Create a new program object.

tered) class the object belongs to.

- Title: The name (title) you want to give to the object.
- Location: The place where you want the object to be created. The place can be the desktop or in a folder. The parameter should be the object Id (See sidebar on Object Ids for explanation) of the location or a fully qualified filename. The location can be either the Desktop or a folder.
- Setup-string (optional): See "Setup String" for a detailed explanation. Some information available in documents in STP sites has been used here.
- The function returns 1 (true) if the object was successfully created or 0 (false) otherwise.

Figure 2 uses sample code to create a new folder object.

The code in Figure 2 creates a folder called MyOwnFolder and places it on the desktop. The location can be changed to any other folder by giving the respective object ID. The object ID of this folder is <MY_OWN_FOLDER>. This object ID is used to create other objects within this folder.

After this is executed, we can see this folder on the desktop with the myownfolder.ico icon. Currently, it is an empty folder. We will add some objects to this folder next.

Note that we can change the object settings for this folder from its Settings notebook.

The code in Figure 3 creates a new program object called MyProgram_in_MyOwnFolder and places it in the MyOwnFolder we created in the previous example. In the setup-string, we have specified the program executable name(mypro-

OBJECT-ID

Object-id is a unique string assigned to a WPS object that can be used to identify it. Some predefined object-ids of system folders include the following.

Object ID	Description
<WP_DESKTOP>	The Desktop
<WP_START>	The Startup Folder
<WP_OS2SYS>	The System Folder
<WP_TEMPS>	The Templates Folder
<WP_CONFIG>	The System Setup Folder
<WP_INFO>	The Information Folder
<WP_DRIVES>	The Drives Folder
<WP_NOWHERE>	The Hidden Folder


```

classname = "WPSHADOW"
title = "MyShadow_in_MyOwnFolder"
location = "<MY_OWN_FOLDER>"

setup-string ="SHADOWID=<WP_EPM>"
rc =
SysCreateObject(classname,title,location,
setup-string)
if rc = 1 then
say " MyShadow : Created"
else
say " Problem in creating
MyShadow. rc = " rc

```

Figure 4. Create shadow objects.

gram.exe) that will run in full-screen mode with the file mydata.dat as a parameter. The object is represented by the myprogram.ico icon inside the folder.

If you are considering placing an object in the Startup Folder (<WP_STARTUP>), make it a shadow of the object.

This piece of code in Figure 4 creates a shadow of the OS/2 EPM editor object inside the folder MY_OWN_FOLDER. Note that the classname is given as WPSHADOW, and in the setup string we have just given the object-id of the EPM editor. To create a shadow of any object in the desktop (through a program), it is necessary to know its object-id.

DESTROYING AN OBJECT

Any existing desktop object can be destroyed using the following SysDestroyObject function.

Syntax : rc = SysDestroyObject(name)
Parameters:
Name: The name of the object to be destroyed. This can be specified object-id or as a fully qualified filename.

Returns: 1 (true) if the object was successfully destroyed or 0 (false) otherwise.

The following code destroys the MyProgram object we created in Figure 3.

```

if SysDestroyObject("<MY_PROGRAM>") then
say "MY_PROGRAM destroyed successfully"
else
say "MY_PROGRAM was not destroyed"

```

Note that we have used the object-id <MY_PROGRAM> to specify the program object. This function destroys the object from the desktop only; it does

SETUP-STRING

The setup-string field used in the SysCreateObject and the SysSetObjectData functions is a set of "keyname=value" pairs separated by semicolons. It defines the behavior of the object. Each object class is expected to document the keynames and associated values that it accepts.

All parameters have safe defaults, so we need not pass any parameter unnecessarily. Also, it shields programmers from knowing all the keynames for an object. Setup string keynames correspond to the settings that can be accessed from the object's Settings notebook pages. The following are some important setup string parameters for some commonly used WPS objects.

For WPFolder Class

KEYNAME	VALUE	PURPOSE
ICONFILE	filename	Sets the object's icon
ICONRESOURCE	id,module	Sets the object's icon "id" refers to the icon resource in the "module" DLL.
MINWIN	HIDE	Hide the views when object is minimized
	VIEWER	Minimize to the minimized window viewer
	DESKTOP	Minimize to the desktop
ICONVIEW	FLOWED	The objects inside the folder are shown using the FLOWED icon view
	NONFLOWED	Shown using the NON FLOWED icon view
	NOLINES	Shown using nongrid icon view.

For WPProgram Class

KEYNAME	VALUE	PURPOSE
EXENAME	filename	The program to execute
PARAMETER	parameters	Parameters to pass to program
STARTUPDIR	pathname	Working directory for the program
PROGTYPE	FULLSCREEN	Run program in OS/2 full screen mode
	VDM	Run program in DOS full screen mode
	WIN	Run program in Win-OS/2 full screen mode
	PM	Run program in PM mode
	WINDOWEDWIN	Run program in Win-OS/2 Window mode
	WINDOWABLEVIO	Run program in OS/2 Window mode
MAXIMIZED	YES	Start program maximized
MINIMIZED	YES	Start program minimized

nothing to the actual files. The object does not exist in the MyOwnFolder anymore.

CHANGING OBJECT SETTINGS

After any object has been created, the object's settings (or the setup data associated with the object) can be changed (or new setup data added) using the SysSetObjectData function:

Syntax: rc=SysSetObjectData(name, setup-string)

The parameters of this function are:

- Name: The name of the object to be destroyed. This can be specified object-id or as a fully qualified file-name.
- Setup-string: A setup-string similar to the one we used in the SysCreateObject function.

The function returns 1 (true) if the object data was successfully updated or 0 (false) otherwise.

Let's say we would now like to associate another icon (mynewicon.ico) with the MyProgram object we created in Figure 3. The following code does exactly that:

```
if SysSetObjectData("<MY_PROGRAM>", "ICON
```

```
PARAM=mynewicon.ico") then
```

```
say "MyProgram setup data updated successfully"
```

```
else
```

```
say "MyProgram setup data was NOT updated"
```

When this code is executed, we can visually see that the icon representing the object gets changed to the mynewicon.ico icon. In a similar fashion, any setup data can be updated (or added) to any existing object.

UTILITIES FOR MANIPULATING WPS OBJECTS

Many utilities exist as freeware or shareware to ease the job of manipulating WPS objects. These utilities (and many more) are available on the Hobbes CD-ROM or by anonymous FTP at ftp.cdrom.com or ftp-os2.nmsu.edu.

- WPSGEN, BUILDSOM: These two programs can be used to create WPS objects from a script file.
- MNREXX: This program creates a new WPS class that gives access to WPS objects in REXX. It has other uses, too.
- WPS Class List: This program comes as part of IBM OS/2 Developer's Toolkit 2.1. It is a WPS class browser

and can be used to create instances of the WPS classes (an interactive way of creating WPS objects).

CONCLUSION

You can programmatically manipulate workplace shell objects from REXX using the functions described here. Applications can use this simple method to extend their functionality to manipulate WPS objects. You can use this to create applications with rich functionality because it lets you create, destroy, or update desktop objects.

Information on WPS programming using REXX is available in the REXX Reference document supplied with OS/2, as part of various WPS utilities, and as separate documentation in many OS/2 FTP sites. (See indexes at the respective FTP sites for a complete list.)

Gowri Sankar Sivaprasad is a graduate student in the Department of Computer Science and a research assistant in the Department of Economics at Iowa State University. He works on OS/2 and UNIX software development and does research in specification languages for object-oriented distributed systems. He can be reached via Internet at sivapras@cs.iastate.edu.



OS/2 Developer

Did You Miss These?

Back Issues

To order your back issues of OS/2 Developer, call now:

800-WANT-OS2
(800-926-8672)

or write to:
OS/2 Developer, P. O. Box 1079
Skokie, IL 60076-8079

Issues are \$9.95 plus \$2.95 shipping and handling.
All orders must be prepaid.

Hurry! Supplies Limited!



REXX and PostScript complement each other. PostScript adds printer formatting, graphics and fonts, and can be easily used from a REXX program. The author of REXX Cookbook shows us how.

By **MERRILL CALLAWAY**

Combining REXX and PostScript

REXX and PostScript are interpreted script languages, complementing each other to a remarkable degree. REXX can construct command strings as well as read and write files, but it cannot perform complex page formatting. PostScript is a page description language capable of complex formats, graphics, and fonts, however, it is clumsy at accessing system resources. This is because the PostScript interpreter resides in the printer and is incapable of responding to a command line interface through the usual parallel port printer connection. By using the strengths of each language, it is possible to code some very useful REXX/PostScript programs for printing to a PostScript printer.

It is simple for a REXX macro to construct and write PostScript command strings directly to the parallel port object or to a print file. As long as PostScript files or strings are written to the logical parallel port (LPT1, LPT2, and so on) to which the OS/2 install program has assigned the PostScript driver, these strings will be interpreted as PostScript commands.

For example, if a Lexmark Optra R laser printer has been installed both with a PCL5 driver assigned to LPT1 and PostScript to LPT2, the program should write PostScript to LPT2. PostScript print files will export to other platforms or to imagesetters for printing. By removing only one line from a PostScript print file, the operator `showpage` (use a text editor), you may then use the file as a "background" page. A REXX program may now append text and/or graphics in PostScript format. A final `showpage` appended to the end of this composite file makes it print the background in addition to the added text/graphics.

HOW DOES POSTSCRIPT WORK?

Like REXX, PostScript uses an interpreter, but it resides inside the printer usually in an EPROM. Just like `CMD.EXE`, the PostScript interpreter waits for an instruction it recognizes and then executes it. When a file is copied or REXX writes one line at a time to LPT1, the PostScript interpreter at the other end of the parallel cable attempts to execute the lines of text as PostScript commands.

PostScript uses a stack in the printer's RAM for holding objects and operators arranged in last in, first out (LIFO) order. PostScript is a combination of `postfix` and `script`. In postfix computing, the operand (the data object) is specified (pushed onto the LIFO stack) before the operator. The PostScript interpreter places objects (including operators) on the stack in the order that it receives them. An operator takes the latest object(s) off the stack and performs a specified operation. An operator might call for the last four data objects on the stack or may need only the latest single object.

All PostScript language objects are ASCII strings and, therefore, easy to construct with REXX. In PostScript, objects can be dictionaries of fonts, definitions of functions, numbers, operators, or literal strings. Literal strings are enclosed in parentheses, which are special characters. The back slash (`\`) is another special character. If a parenthesis or a back slash is to be included as part of a literal string, it must be escaped by prefixing it with a back slash. A PostScript program to print the string `roast beast` looks like this:

```
/Courier findfont 10 scalefont setfont
(roast beast)
72 720 moveto show
showpage
```



```

/* PE.CMD Envelope PS Printer */

/* NOTE: Change the following value to match your type of
** printer. If envelope is center fed; e.g. Epson EPL-7500,
** then change to center=1. Lexmark Optra R is side feed.
** Side feed envelopes have different x and y values.
** Different values are accounted for in xadd and yadd.
*/

center=0

IF center=1 THEN DO
  /* Center feed envelopes. */
  xadd=0
  yadd=0
END
ELSE DO
  /* Optra R side feed envelopes. */
  /* Different side feed printers may need different values. */
  /* Print full sheet of paper first to verify dimensions. */
  xadd = -108
  yadd = 158
END

START:
drop match
print='no'
savefile=''
/*
** NOTE: Change the following database file
** to a filespec of your choosing.
** The format of the file uses '|' as a separator.
** Each line begins with '|' and '|' separates each
** line of the address. The address keyword is
** the first entry. Example:
** |keyword|line1|line2|line3|line4|
** There may be as many lines as necessary, but each
** one must be separated by '|'.
*/
addrfile='C:\REXX\ADDENV\ADDFILE.TXT'
SAY 'Enter address keyword, OR [Enter] for new address.',
    'Q=Quit. L=List Addresses.'
PARSE UPPER PULL input
IF input == 'Q' THEN EXIT 0
IF input == 'L' THEN DO
  DO i=1 WHILE LINES(addrfile) \= 0
    line.i=LINEIN(addrfile)
    PARSE VAR line.i '|' name.i '|' .
    SAY name.i
    /* OS/2 Command Window only holds 23 lines...*/
    IF i//23=0 THEN DO
      SAY ,
      '23 lines visible. Press [Enter] to see next 23 lines.'
      PULL .
    END
  END
  CALL STREAM addrfile, 'C', 'CLOSE'
  SIGNAL START
END
SIGNAL ON HALT
IF input == '' THEN DO
  SAY 'Enter address: line 1 [Enter], line 2 [Enter], etc.'
  SAY 'Enter [Ctrl-C] AFTER last [Enter] to stop.'
  DO m=1
    PARSE PULL add.m
  END m
END /* input='' */
IF input \= '' THEN DO
  IF LINES(addrfile) THEN DO
    DO i=1 WHILE LINES(addrfile)

```

If the above were saved to a file, you could copy it to LPT1 and it would print the string `roast beast` in Courier font, 10-pt size, and at the top of the page with 1-inch top and left margins. The PostScript interpreter puts `/Courier` on the stack (a / causes PostScript to interpret a string as the name for an internal object). The operator `findfont` finds the Courier font inside the printer's memory of resident fonts. A number 10 is pushed on the stack, and a `scafont` operator takes the two operands—the result of `findfont` and the scale number—and scales the entire font. The operator `setfont` sets up the font dictionary using the information from the stack (the scaled font) for our program to use. The actual PostScript object now on the stack is really an address pointer to what is called an encoding vector, which is itself a lookup table of codes and corresponding printable characters. By means of this encoding vector, PostScript automatically builds each character internally in software when `setfont` is interpreted.

The parentheses tell PostScript that the next line is a literal string, so it pushes it onto the stack (on top of the font dictionary of 10-pt Courier). Note that pushing onto the stack is equivalent to writing a string to the parallel port since the PostScript interpreter is waiting at the other end.

On top of the font and the literal string, the PostScript interpreter now pushes the X coordinate measured in points (there are 72 points in an inch), 72, followed by the Y coordinate, 720. Points are units of measure persisting from the age of printing with lead type. PostScript measures its coordinates from an origin (0,0) that begins at the lower left corner of the paper and moves up in the Y direction and to the right in the X direction. The coordinates represent the upper left hand of a letter-size paper with top and left margins of one inch. This is where the string literal will print.

The operators, `moveto` and `show`, are pushed onto the stack. `Moveto` moves to the current point on the page determined by the two underlying X,Y coordinates on the stack. `Show` commits to print the text underlying it at the current point on the page. No text is actually printed, however, until `showpage`. Think of an invisible cursor moving around the page according to the `moveto` operators. The `show` operator commits to text placement at each new location. After everything is complete (the font, the text, and where it

Figure 1. PE.CMD REXX/PostScript Print Envelope Utility (continued on page 19).

goes), the last PostScript operator, **showpage**, prints the page as it was described. The **showpage** operator is always the last PostScript operator on a page.

CUSTOM FORMS

By using PostScript, it is possible to make a REXX program that custom prints complex forms! The technique is simple. In a DTP or word processor, lay out the blank form. Print a copy. Also, save the PostScript print file. Edit the print file and remove *only* the line **showpage** near its end. Write a REXX macro to obtain user input data.

With a millimeter ruler, measure the X,Y coordinates for placement of each line of this data text within the blank form. Multiply millimeter measurements by 2.83465 to obtain points. Use REXX to construct an appropriate PostScript print file like the previous example, placing the user data within the form. In the same REXX program, copy the blank form print file (minus **showpage**) to the logical port assigned to PostScript (LPT1 and so on), followed by the REXX-constructed, data-printing PostScript code. The single, final **showpage** (from the REXX program) will print the whole page, complete with data. Using VX-REXX, a user-friendly interface could be developed and the most complex custom forms printed with very little programming overhead. Now for some specific REXX/PS code.

PRINTING ENVELOPES

It takes far too long to load a DTP or word processor program just to print one envelope. This calls for a REXX and PostScript envelope print utility, which runs quickly from the OS/2 Window. It not only prints envelopes but also keeps a simple and efficient database of addresses too.

Users can copy an address from an open document directly into the OS/2 Window via the clipboard. This utility is quicker to use than a DTP or word processor program, even when they are open. After a few addresses are saved in the database, the most frequently used addresses will be just two or three keystrokes away.

PE.CMD works with either a center-feed printer (such as the Epson EPL-7500) or a to-one-side envelope-feeding printer (Lexmark Optra R). It is easily adapted to work with any PostScript printer capable of feeding standard No. 10 business-size envelopes of 9 1/2 x 4 1/8 inches (you

```

line.i=LINEIN(addrfile)
PARSE VAR line.i '|key|' line.i
IF input=key THEN DO
  match=1
  DO m=1 WHILE line.i \= ''
    PARSE VAR line.i add.m '| ' line.i
  END m
  LEAVE i
END /* if input...*/
END i
IF SYMBOL('match')='LIT' THEN DO
  SAY 'No match found. Try again...'
  CALL STREAM addrfile, 'C', 'CLOSE'
  SIGNAL start
END
DO j=1 TO m-1
  SAY add.j
END j
CALL STREAM addrfile, 'C', 'CLOSE'
END /* if lines() */
ELSE DO
  SAY 'Could not open your address file. Try again.'
  SIGNAL start
END /* ELSE */
END /* input \= '' */
HALT:
m=m-1
SAY '[P]=print; [S]=save address; [B]=do both. [Q]=quit.'
PARSE UPPER PULL answer
IF answer == 'Q' THEN EXIT 0
IF answer == 'S' | answer == 'B' THEN DO
  SAY 'Enter keyword for address.'
  PULL savekey
  savekey='|savekey|'
  DO s=1 TO m
    savekey=savekey||add.s'|'
  END
  rc=LINEOUT(addrfile,savekey)
  IF rc=0 THEN SAY 'Saving address and keyword...'
  CALL STREAM addrfile, 'C', 'CLOSE'
  '@SORT < 'addrfile
END /* S | B */
IF answer == 'P' | answer == 'B' THEN
  CALL printaddress m, add., xadd, yadd
  SIGNAL start
EXIT 0

/* internal function */
printaddress: PROCEDURE EXPOSE m add. xadd yadd

/* postscript commands and parameters */
font=,
'/NewCenturySchlbk-ISOLatin1 findfont 12 scalefont setfont'
coordx=410 + xadd /* left margin */
coordy=300 + yadd /* top margin */
pscommand='moveto show'
pshow='showpage'
tran='0 -612 translate'
rotate='90 rotate'

SAY 'PRINTING...'

/*
** NOTE: With the Optra R printer, there can be logical
** parallel port devices, LPT2, LPT3, etc. A normal
** installation with both PCL5 and PostScript PS level 2,
** would assign PostScript to logical device LPT2.
** Change the following to whatever parallel device name
** your PostScript printer is connected.

```

Figure 1. PE.CMD REXX/PostScript Print Envelope Utility (continued on page 20).


```

** To print to a file, put a different value for par
** (a filespec).
*/

par='lpt2'
/* re-encode to ISOLatin vector */
CALL LINEOUT par,'/NewCenturySchlbk-Roman findfont'
CALL LINEOUT par,'dup length dict begin'
CALL LINEOUT,
  par,'{1 index /FID ne {def} {pop pop} ifelse}forall'
CALL LINEOUT par,'/Encoding ISOLatin1Encoding def'
CALL LINEOUT par,'currentdict'
CALL LINEOUT par,'end'
CALL LINEOUT,
  par,'/NewCenturySchlbk-ISOLatin1 exch definefont pop'
/* end of re-encoding vector */
CALL LINEOUT par,font rotate tran
DO i=1 TO m
  CALL LINEOUT par,'('add.i.)'
  CALL LINEOUT par,coorx coorx pscommand
  coorx=coorx-12
END
coorx=135 + xadd
coorx=418 + yadd
CALL LINEOUT par,'(WHITESTONE)'
CALL LINEOUT par,coorx coorx pscommand
coorx=coorx-12
CALL LINEOUT par,'(511-A Girard Blvd. SE)'
CALL LINEOUT par,coorx coorx pscommand
coorx=coorx-12
CALL LINEOUT par,'(Albuquerque, NM 87106)'
CALL LINEOUT par,coorx coorx pscommand
coorx=coorx-12
CALL LINEOUT par,'(USA)'
CALL LINEOUT par,coorx coorx pscommand
coorx=coorx-12
font='/Helvetica-Narrow-Bold findfont 30 scalefont setfont'
CALL LINEOUT par,font
coorx=175 + xadd
coorx=275 + yadd
CALL LINEOUT par,'(AIR MAIL)'
CALL LINEOUT par,coorx coorx pscommand
CALL LINEOUT par,pshow
CALL STREAM par,'C', 'CLOSE'
RETURN

```

Figure 1. PE.CMD REXX/PostScript Print Envelope Utility (continued from page 19).

may alter the code for a different size envelope or multiple choices for envelopes). Figure 1 is a listing of PE.CMD.

PROGRAM DESIGN

The program is in three sections. First, it maintains the address database file and allows the user to enter new addresses or call up frequently used addresses by entering a keyword. The database file uses a separator character to begin and end each line of the address. I used a pipe symbol. The keyword for the address is a mnemonic string and may be input to recall an entire address. Each database entry begins with the keyword for the address, and the address takes up exactly one line in the database file.

Printers feed envelopes through in at least two ways: centered and to one side. If the printer feeds envelopes at the center of the paper tray, then code center=1, and xadd=0; yadd=0. In this listing, Center=0 and xadd, yadd are set for No. 10 business envelopes feeding into a Lexmark Optra R. It feeds envelopes at the right side of the paper path from the rear input tray. Optra R users should also turn off the AES program. Auto software selection is not needed for directly copying PostScript to LPT2.

The repeatable section labeled START: allows data entry, recall of an address previously saved, or viewing of the address keyword list. In case a fresh address is entered line by line,

Ctrl+C is used to signal HALT to escape data entry mode.

The final section is a PROCEDURE-labeled printaddress, which the program calls when it needs to print the address on an envelope. Note the list of exposed variables. There are many similarities between this internal function and the short sample PostScript program. Note the use of REXX expressions and tokens to build up the lines of PostScript data and operators. To view the PostScript output of this program, comment out one line, par='LPT2' and substitute par=<filespec>. Figure 2 shows a typical print output file.

REENCODING A VECTOR

The program reencodes the character table to the ISOLatin1 vector. This makes a different encoding vector for characters in the chosen font, NewCenturySchlbk. The new encoding enables printing of accented characters. The normal characters are unaffected, so if you have no German addressees, you will never notice this feature. The reencoding PostScript instructions do the following:

```

/NewCenturySchlbk-Roman findfont
dup length dict begin
{1 index /FID ne{def}{pop pop}ifelse}forall

```

1. Find the font and make a copy of the font dictionary. Copy all entries {def} except those equal to font ID /FID; ne means not equal to; pop means to raise the stack and discard the top item; ifelse and forall are loop control, and 1 index is the iteration specifier. The brackets {} organize the grouping of items.

```

/Encoding ISOLatin1Encoding def
currentdict
end

```

2. Replace the font's Adobe StandardEncoding vector with the ISOLatin1Encoding vector, which places accented characters in control character positions (which are not normally used by fonts). ISO-Latin1Encoding is a built-in 256-element array of character names defined in the PostScript systemdict, a part of the interpreter.

```

/NewCenturySchlbk-ISOLatin1 exch
definefont pop

```

3. Give the new font a different name to reflect the fact that it is reencoded with ISOLatin1Encoding.


```

/NewCenturySchlbk-Roman findfont
dup length dict begin
{1 index /FID ne {def} {pop pop} ifelse}forall
/Encoding ISOLatin1Encoding def
currentdict
end
/NewCenturySchlbk-ISOLatin1 exch definefont pop
/NewCenturySchlbk-ISOLatin1 findfont 12 scalefont setfont
90 rotate 0 -612 translate
(Watcom International Corporation)
302 458 moveto show
(415 Philip Street)
302 446 moveto show
(Waterloo, Ontario)
302 434 moveto show
(CANADA N2L 3X3)
302 422 moveto show
(WHITESTONE)
27 576 moveto show
(511-A Girard Blvd. SE)
27 564 moveto show
(Albuquerque, NM 87106)
27 552 moveto show
(USA)
27 540 moveto show
/Helvetica-Narrow-Bold findfont 30 scalefont setfont
(AIR MAIL)
67 433 moveto show
showpage

```

Figure 2. PostScript Output from PE.CMD.

Now that the new font is defined, the program can find it, scale it, and set it as usual. PostScript Language Reference Manual, page 275f and Appendix E provides more on encoding vectors.

LANDSCAPE PRINTING

It is now necessary to transform the coordinate system lengthwise, the way envelopes feed into PostScript printers. The default origin (0,0) is at the lower left corner of the page, but the coordinate system may be rotated and translated to suit. Think of a piece of paper on your desk upon which you place your right hand, palm up, with your index finger along the left, long side (the Y axis) and your thumb along the lower, short side (the X axis). Your hand represents the default coordinate system, called a right hand coordinate system. The PostScript operator `rotate` turns the entire coordinate system (your hand with the thumb and index finger at right angles) around the present origin counterclockwise the number of degrees specified by the number placed on the stack just beneath the `rotate` operator.

The positive quadrant of X and Y values rotates off the page during a 90 `rotate` operation. Therefore, we need to complete a coordinate translation

before our paper is lined up correctly for landscape printing. PostScript shifts the coordinate system using the `translate` operator, which uses the values of X and Y on the stack underneath the operator. The sequence of PostScript objects: 0 -612 `translate` moves no distance along the X axis and minus 612 points (8.5 inches) along the Y axis. (Note that the shift is negative along the newly rotated Y axis.) The PostScript measurements are still along the index finger for the Y coordinates and along the thumb for X coordinates. The positive direction is toward the fingertips. The landscape transformation (90 `rotate` followed by 0 -610 `translate`) effectively prints everything at 90 degrees to the original orientation of the paper.

PE.CMD builds the PostScript strings and then writes them to the parallel port using the REXX function `LINEOUT()`. The parallel port can be treated as a stream. Because PostScript is an interpreted language, these PostScript code strings are executed as soon as an operator string arrives at the printer. Nonoperator, or data, strings are pushed onto the stack as they arrive.

TEXT POSITIONS ON ENVELOPES

Determine coordinates for the text po-

sitions on an envelope by measuring a regular sheet of paper with a standard No. 10 business envelope oriented lengthwise and centered or positioned to one side depending on your printer. The printer always thinks it is printing on a standard-size page, so we must make sure to print only in the area covered by the envelope. Figure margins are based on the standard $8\frac{1}{2} \times 11$ area. It is always a good idea to print on a full sheet of paper until the printing area lines up correctly.

The final section of the print function hard codes the return address in the upper left corner of the envelope. Note how to move down one line by subtracting 12 points from the Y coordinate. The final command is the PostScript `showpage`. PostScript and REXX seem to have been made for each other!

Merrill Callaway is the author of two REXX books, *The AREXX Cookbook* and *The REXX Cookbook*, tutorial guides to Amiga REXX and OS/2 REXX, respectively. Before he became a full-time author and AREXX columnist, he was a senior logistics engineer in the aerospace industry, where he pioneered the computer automation of Logistics Support Analysis Records for avionics equipment. He holds degrees in applied mathematics and art from Brown University in Rhode Island. He may be reached at callaway@indirect.com on the Internet.

REFERENCES

PostScript Language Reference Manual, 2d ed., Addison-Wesley, 1990. (ISBN 0-201-18127-4)

PostScript Language Tutorial and Cookbook, Addison-Wesley, 1985. (ISBN 0-201-10179-3)

Callaway, Merrill. *The REXX Cookbook*, Whitestone, 1995. (ISBN 0-9632773-4-0)



This article is reprinted with permission from Inter Rexx Newsletter, Vol. 2 Issue 1, January 1995). Copyright 1994-1995 Markus Pelt-Layman. All rights reserved.

By MARKUS PELT-LAYMAN

PDS Transfers Using EHLLAPI in REXX

The Emulator High-Level Language Application Programming Interface (EHLLAPI) running under OS/2 Communications Manager provides users the capability to write client/server applications using emulated IBM 3270 terminals connected to IBM mainframes. Although EHLLAPI supports the Basic, C, COBOL, and Assembler languages, the power and flexibility of REXX significantly reduce programming effort and speed up development.

EHLLAPI is available from almost every vendor that supports 3270 terminal emulation over 3270 emulator adapter cards, SDLC adapter cards, or LAN adapter cards under both OS/2 and DOS/Windows. (The RLIM.ZIP shareware package, available on the Rexx-R-Us BBS at (303) 4401351 and on other BBSs provides EHLLAPI support specifically for Enterprise REXX and Quercus REXX under DOS and Windows. However, the calling syntax is slightly different than that for OS/2.)

THE 3270 WORLD

The IBM 3270 family of terminals was already widely used before the advent of the personal computer. A typical local configuration (shown in Figure 1) consists of an IBM 3274 control unit (CU) connected via a channel to an IBM 370/390 mainframe. The control unit can connect up to 32 IBM 3270-type terminals via coaxial cable in a star configuration. Distances between CU and mainframe as well as between CU and terminal are limited without additional equipment.

For remote connections, a different control unit is used, but the same 3270-type terminals can be attached to it, again using coaxial cable in a star configuration. The

control unit in this case is connected using modems and phone lines to a Front End Processor (FEP), which is attached via a channel to the mainframe.

Terminals come in two basic flavors: the older, dumb Control Unit Terminals (CUT), which rely heavily on processing logic in the CU, and the newer, smart Distributed Function Terminals (DFT), which off-load a lot of the terminal processing from the CU. 3270-compatible terminals come in many configurations and with many options, such as APL and Katakana keyboards, security card readers, as well as light pens, graphics, and IBM 328X printer support.

While the 3270 terminal and control unit market was lucrative for both IBM and many compatible vendors, the introduction of the IBM PC saw the rise of 3270 emulation adapter cards. The initial problem for many large companies was that every desk that once had a 3270 terminal on it now had a separate 3270 terminal and a personal computer (which didn't even talk to the mainframe). The solution was a 3270 emulation adapter card, which allowed a PC to act as a real 3270 terminal by attaching via coax cable to a CU, eliminating the need for a standalone terminal.

While the adapter card provided the low-level hardware compatibility to talk to the control unit, the PC required software to actually display output from the mainframe on the PC screen as well as accept keystrokes and send them to the mainframe. This emulation software was usually bundled with the adapter card and often specific to the adapter card. Each vendor had his or her own way of doing things.

3270S IN A LAN WORLD

The introduction of LANs (Local Area

Networks) created yet another problem. PCs that were already connected to the mainframe using 3270 emulation adapter cards needed a LAN adapter card to connect to the LAN. In large corporations, this meant having each PC wired using two coaxial cables: one for the LAN and one for the mainframe. The solution to this double wiring problem was to connect to the mainframe over the LAN somehow, thus eliminating the 3270 emulation adapter card and coax cable. Of course, this introduced even more ways to provide 3270 emulation, again with each vendor doing things in nonstandard ways.

EHLLAPI was the answer to standardizing the way in which PC applications could programmatically address 3270 emulation terminals, regardless of the physical connection from PC to mainframe. Under OS/2 Communications Manager, this is especially obvious when one considers that OS/2 supports 3270 sessions using 3270 emulation adapter cards, Ethernet adapter cards, Token-Ring adapter cards, SDLC adapter cards, and even async serial adapter cards, as well as many others.

Regardless of the hardware and software configuration, EHLLAPI provides the same interface to all.

LU 6.2/APPc VS. 3270 EMULATION

Some people say 3270 emulation is old technology and that it will go the way of the dinosaur. IBM is pushing a new protocol called LU 6.2 (also known as APPC or Advanced Program-to-Program Communications) as the protocol of choice (the 3270 terminal protocol is referred to as LU2).

While it is true that LU 6.2 is better designed and allows far more flexibility than what is provided with 3270 terminal emulation, it does require that mainframe applications be rewritten to specifically use it. From a mainframe application point of view, it is still far easier to write an application for a 3270 terminal than it is to write one for APPC. In addition, many legacy systems would require major program logic changes to accommodate APPC.

This is partially due to the fact that existing 3270 terminal applications are encapsulated by a terminal monitor program such as CICS or TSO, which is responsible

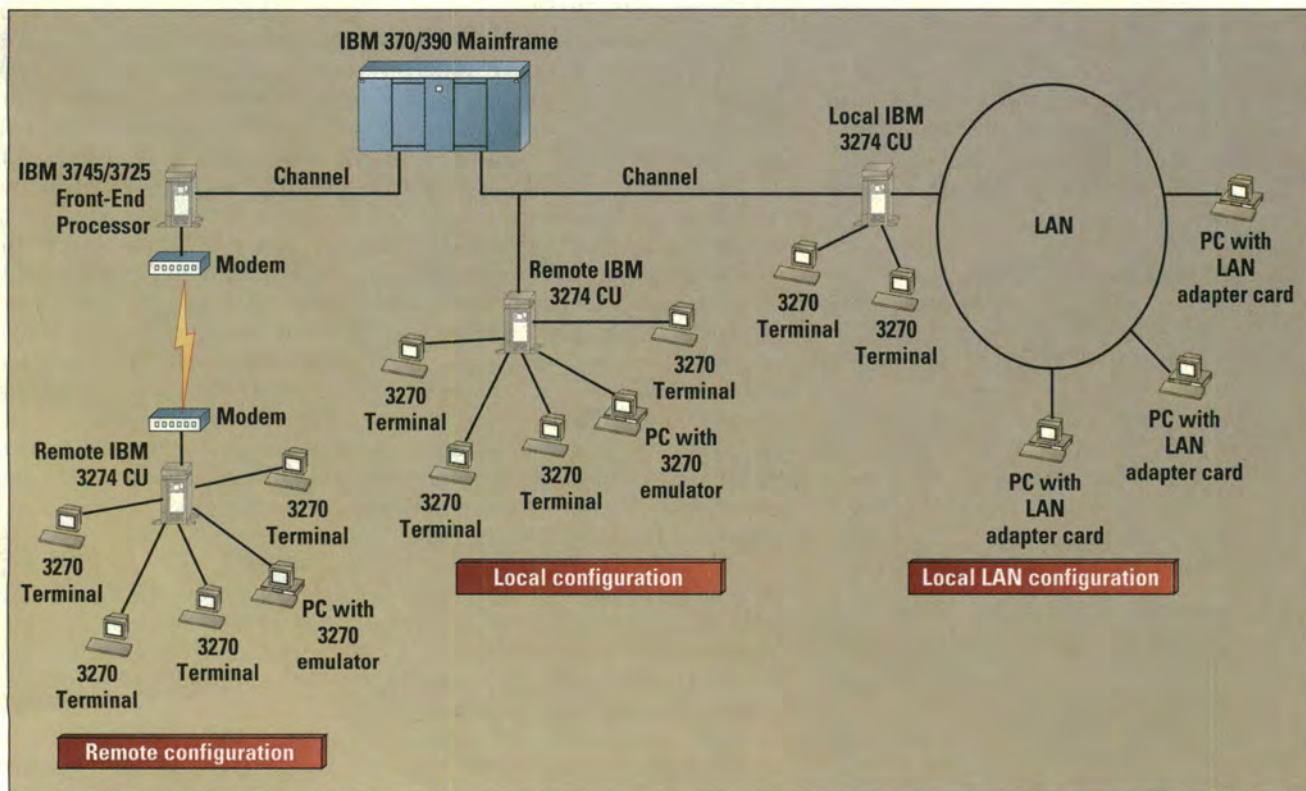


Figure 1. Sample 3270 configurations.

for much of the VTAM network protocol establishment and security authentication processing. Until all mainframe applications are converted to use APPC, many legacy systems using 3270 terminals will remain. 3270 terminal emulation may eliminate 3270 standalone terminals but not 3270 mainframe applications. What EHLLAPI can provide right now is a way of interfacing with such legacy systems and possibly even putting a new coat of GUI paint on an old 3270 character-based mainframe application.

THE 3270 TERMINAL

The original 3270 terminal consisted of a green monochrome monitor and a keyboard. The keyboard had a standard QWERTY layout with 12 additional program function keys (PF keys) for special functions on the right-hand side where the numeric keypad usually appears on personal computer keyboards.

The monitor displayed a character-based (as opposed to graphics) screen 24 lines by 80 columns, with an additional status line at the bottom of the screen (called the Operator Information Area, or OIA) reserved for terminal session indicators (such as shift state, keyboard locked state, and so forth). While the original 3270 models supported only limited display attributes for each character, such as low or high intensity or invisibility, the later models support different color and extended attributes such as reverse video and user-defined character sets (called Programmed Symbols in 3270 parlance) as well as different screen sizes.

FIELDS AND ATTRIBUTES

What is displayed on the screen is determined by what is placed in the terminal's presentation space (PS) buffer. In its simplest form, the PS buffer acts like a PC video display buffer. For example, a 24-line-by-80-column screen has an associated 1,920-byte ($=24 \times 80$) PS buffer. To display the character "A" on line 1 column 1 of the screen, place the byte

red	high-intensity input field
green	low-intensity input field
blue	low-intensity output field
white	high-intensity output field

Table 1. Basic color support.

"A" in the first buffer location. The buffer address for the character at line 2 column 1 is 80 (or offset 79).

Aside from straight text bytes, you can also place attribute bytes in the buffer. A buffer location set with an attribute byte marks the start of a new field. The value of the attribute byte describes some of the basic attributes for all characters that follow in the field up to the next attribute byte (the next field). Attribute bytes themselves display on screen as a blank (which cannot be modified via the keyboard).

For example, a bit in the definition of an attribute byte indicates whether the field should be displayed in high or low intensity. Another set of bits describes whether the field is an input field or output only field (not modifiable by the keyboard user). One more bit is reserved to indicate if the field has been modified (MDT or Modified Data Tag bit). This last bit is especially useful when reading the PS buffer, as it indicates a field that has been changed in some way (this bit can also be set by programs to simulate user modifications).

The PS buffer wraps at the end of the screen: a field started by the last attribute byte will continue from line 24 column 80 to line 1 column 1 unless an attribute byte is put in line 1 column 1 or line 24 column 80.

EXTENDED ATTRIBUTES

To support more than the basic attributes previously described (intensity, input/output, and modified), three additional buffers are used that are conceptually similar to the PS buffer in addressing. These extended attribute buffers are used for:

1. Extended highlighting (reverse video, blinking, and underlined)
2. Extended colors (blue, red, pink, green, turquoise, yellow, and white)
3. Programmed Symbols (usually used for graphics).

Basic color support does not use extended attributes. A simple translation scheme using the basic field attributes is used as shown in Table 1

ATTENTION KEYS

Following is the usual cycle for interaction on a 3270 terminal:

1. A mainframe program displays output on screen and waits for user input
2. The user types information in input fields displayed on screen and presses an attention key.

The user signals that he or she is done typing input into fields by pressing one of the attention keys. All keys other than attention keys are handled locally, and no data is transferred to the mainframe until an attention key is pressed. The following are the attention keys:

- Enter key
- Clear key (also clears screen)
- PA1, PA2, and PA3 keys
- PF1 through PF24 program function keys
- Test Req key.

In addition, certain terminal attachments, such as a card reader, magnetic slot reader, hand scanner, or light pen, can also signal attention.

MAINFRAME PROGRAMMING

Under MVS TSO you can use the TPUT (Terminal PUT) and TGET supervisor call instructions/macros to send and receive 3270 data streams respectively. At a higher level, the panels used by ISPF and the screen definitions used by CICS are translated to 3270 data streams.

3270 DATA STREAMS

From the point of view of the mainframe, the 3270 terminal is driven by sending it commands. The basic commands are:

- Write
- Read buffer
- Read Modified.

When a mainframe application sends the 3270 terminal a Write command, it is followed by a sequence of data characters mixed with orders. There is a concept of current buffer address (or just buffer address for short), which is incremented whenever a data character is encountered in the write data stream (after the character is put into the PS buffer at the current buffer address). The orders encountered in a write stream can manipulate the PS buffer as well as alter the current buffer address. The changes made by the write stream to the PS buffer in turn change what is displayed on the screen of the terminal.

ORDERS

Table 2 shows the various orders that can be included in a 3270 data stream and the effect each has on the PS buffer and current buffer address.

When a Read Buffer command is sent by the mainframe to the 3270 terminal, the terminal transmits back to the host the entire PS buffer. The format of the returned data is data char-



acters mixed with SF orders for each field attribute byte (so the mainframe application can decipher where each field started).

The Read Modified command works similarly but returns data and SF orders for only modified fields (fields with the MDT bit on in their basic attribute byte.) Either the MDT bit was turned on by the user making modifications to the field using the keyboard, or the mainframe application sent the attribute byte originally with the MDT bit already on. The MDT bits are reset once all modified fields have been transmitted. The Read Modified command also adds SBA orders before every SF order to indicate where each modified field started.

The data stream returned by Read and Read Modified commands is preceded by three bytes, including the AID (Attention Identification) character, which identifies the key the user pressed to cause the Read or Read Modified to be completed.

PC PROGRAMMING—EHLLAPI

While 3270 data streams are used to access the 3270 terminal PS buffer from the mainframe, EHLLAPI is used to access the emulated terminal's PS buffer as well as send keystrokes from the PC side. The basic idea behind EHLLAPI is to send key-

strokes and read the screen (that is, access the PS buffer) to emulate what a human user would do manually.

The EHLLAPI specification provides some 50-plus functions. Luckily, only a handful are needed for most applications. Before discussing the most often used EHLLAPI functions in detail, let's digress to the problem that triggered the writing of this article.

PC/MAINFRAME FILE TRANSFER

Both MVS TSO and CMS include a command called `IND$FILE`, which can be used to send or receive single files from a 3270 emulator to the mainframe. The emphasis here is on the words single file (although the newest VM/CMS version of `IND$FILE` does allow multiple files using wildcard characters). Most terminal emulator products (including OS/2) provide a `SEND` and `RECEIVE` command on the PC side to initiate a file transfer to or from the mainframe, respectively:

```
IND$FILE GET hostfile options
```

or

```
IND$FILE PUT hostfile options
```

The `SEND` and `RECEIVE` commands themselves issue an `IND$FILE` command to the mainframe host under

SBA	Set Buffer Address order changes the current buffer address to that specified in the order
RA	Repeat to Address order fills the PS buffer with the character specified in the order from the current buffer address to the one specified in the order.
SF	Start Field order inserts a basic attribute byte specified in the order into the PS buffer at the current buffer address. This starts a new field.
SFE	Start Field Extended order is similar to SF order except the order may include both a basic attribute byte as well as extended attribute bytes.
MF	Modify Field order modifies basic and/or extended attributes for the field that starts at the current buffer address
SA	Set Attribute order changes the extended attributes for a character at the current buffer address
PT	Program Tab order changes the current buffer address to the next input field (starting from the current buffer address)
EAU	Erase All Unprotected order sets all input fields to nulls from the current buffer address to that specified in the order.

Table 2. 3270 Data Stream orders.

Simple Database Management

dbfREXX Version 2

Now you can have simple database management from your REXX programs, without spending a lot of money!

dbfREXX extends REXX with functions to read, write, and create database files and indexes.

Files supported are:
DBF, DBT, FPT,
NTX, & NDX.

\$ 99

Visa and MasterCard

dSOFT Development Inc.

713-537-0318

Voice & Fax

the covers, so the user is never really aware of the command syntax of the `IND$FILE` host command. If you want to check if you have `IND$FILE` on your host, simply try the `IND$FILE` command without any operands from a 3270 session window. An indication that the command was not found probably means `IND$FILE` is not available.

The syntax of the `SEND` and `RECEIVE` OS/2 commands is fully explained in the OS/2 online help (issue `HELP SEND` or `HELP RECEIVE` from an OS/2 command session). In brief, the syntax is:

```
SEND pcf file [session:] hostfile
[options]
```

and

```
RECEIVE pcf file [session:] hostfile
[options]
```

where all lowercase items should be replaced with actual values and brackets indicate optional items (do not enter the brackets themselves). Check the OS/2 online help for the syntax details.

During the actual file transfer, the session window usually goes blank. This is to prevent the user from becoming confused if he or she sees the file transfer data displayed (which may include nonviewable characters if binary data is transmitted).

PC VS. MAINFRAME FILES

PC files come in two basic flavors: ASCII text files and proprietary binary data files. Mainframe files are distinguished by file organization, record format, record length, and block size. During file transfer, differences in file structure and contents may have to be resolved. Following are the major problems:

- ASCII versus EBCDIC encoding of text data. The PC uses ASCII encoding for text characters, and the mainframe uses EBCDIC encoding. This means that whereas the letter "A" is represented in ASCII as a byte whose value is 65, it is represented in EBCDIC as a byte with the value 193. Obviously, without translation, an ASCII file will be printed or displayed as garbage on a mainframe. Similarly, a mainframe file containing names and addresses will look like garbage when typed or printed on a PC without translation.

The `SEND` and `RECEIVE` commands (and `IND$FILE`) can translate ASCII to EBCDIC and vice versa when

the option "ASCII" is added to the end of the command.

- CR/LF translation to records and vice versa. For text files on the PC, a line or record is marked by the presence of a carriage return and/or line feed character. On the mainframe, text files do not include these control characters. Instead, records in a file are all the same length (`RECFM=F`) or variable with a 4-byte record descriptor word preceding each record indicating the length of the record that follows. Clearly, when receiving a mainframe file without CR/LF control characters, the file is not readable on the personal computer. Conversely, the presence of CR/LF data will display as garbage on the mainframe and possibly cause problems with lines being split or running together erroneously.

Fortunately, the "CRLF" option on the `SEND` and `RECEIVE` commands will translate records for text files correctly in this case.

- Record and block descriptor words for variable-length mainframe files are always stripped when sent to the PC by `IND$FILE`. This in effect means that variable-length files are unusable *unless* they are text files and you use the "CRLF" option.
- Little-Endian vs. Big-Endian problems can occur for binary files that include numeric values. The mainframe stores numeric values larger than a byte in most-significant-byte-first sequence. For example, the value 256 is stored in a file on the mainframe as two bytes '01'X and '00'X. On the PC, the least-significant bytes are always stored first (for example, '00'X '01'X means the value 256).

Unfortunately, there is no way to know which values would require switching bytes around without knowledge of a binary file's record layout and structure. `IND$FILE` will not provide any assistance for this problem. You may need to massage the file after transfer to make it usable on the target system in this case.

- File naming problems may occur. Since PC file names have an 11-character (8.3) format and mainframe PDS member names have only eight characters, something will have to give. On the PC side, the three-letter filename extension usually indicates file type, which on the mainframe is normally indi-

cated by the last qualifier of the data set name. For example:

```
C:\MYPROJ\MYFILE.ASM
```

could be translated to:

```
'MYUSRID.MYPROJ.ASM(MYFILE)'
```

However, even with such a translation scheme, naming conventions for member names are stricter than those for PC file names (which may include numerics as the first character, as well as special symbols not allowed for member names).

PARTITIONED DATA SETS

MVS supports a multitude of file organizations, one of which is called the Partitioned Data Set (PDS). A PDS is a collection of homogeneous sequential files. All the member files (or members) must share the same logical record length (`LRECL`), block size if blocked (`BLKSIZE`), and record format (`RECFM=F` for fixed-length records or `RECFM=V` for variable-length records). With the exception of the homogeneity requirement, PDSs are similar to PC directories. While the mainframe has utilities to copy and manipulate entire PDSs and the PC has utilities to do the same for directories, the `IND$FILE` file transfer program has no direct support for PDS files.

To transfer an entire PDS, each member will need to be sent individually (that is, a `SEND` or `RECEIVE` command must be issued for each member separately). This is a minor inconvenience for transferring a small number of members, but it is a show-stopper if you have to transfer a PDS with a hundred or more members. The solution, of course, is to write a small REXX program to issue the commands for us.

SPECIFICATIONS

Ideally, we would like a command to upload a PC directory to a mainframe PDS and another command to download a mainframe PDS to a PC directory. For the upload to a PDS to work, all files in the source PC directory must share the same file characteristics (remember the homogeneity requirement of PDSs). In addition, we should probably allow for the specification of the 3270 session to perform the transfer as well as the option to pass the CRLF and ASCII options for text files. So the full syntax becomes:

```
UPPDS pcdir pdsname [session:] [options]
```


DOWNPDS pdsname pcdir[session:] [options]

Both the pcdir operand in the UPPDS command and the pdsname operand in the DOWNPDS command support wild card characters (? for single character match and * for multiple character match) to specify that only those files or members that match the wild card specification be transferred. For example:

```
DOWNPDS 'SYS1.MACLIB(IEF*)' C:\MACS
```

will only download members whose names start with the prefix IEF.

EHLLAPI APPLICATION DESIGN

As previously stated, the basic idea behind EHLLAPI is to send keystrokes and "read" the emulated terminal's screen, that is, to simulate what a user would do manually. A good starting point for design, therefore, is to write down what exactly a user would do manually.

To upload files from a PC directory to a mainframe PDS:

1. The user issues a DIR command from an OS/2 command window to get the names of all the files in the PC subdirectory to be uploaded.
2. The user writes down on a notepad a list of all the files to be uploaded.
3. The user makes sure that he or she has a 3270 session with the target host.
4. The user checks that the target PDS exists using the TSO LISTDS command and, if not, creates it on the mainframe using the TSO ALLOC command.
5. The user issues a SEND command from an OS/2 command window for each file in the list and checks to make sure each upload finishes successfully.

To download members of a PDS to a PC directory:

1. The user makes sure that he or she has a 3270 session with the target host.
2. The user issues a LISTDS TSO command to get a list of all member names in the PDS.
3. The user writes down a list of all the members to be downloaded.
4. For each member in the list, the user issues a RECEIVE command from an OS/2 command window and checks to make sure each download finishes successfully.

COMMAND REQUIREMENTS

Aside from using the SEND and RECEIVE

OS/2 commands to perform the transfer of individual members, we will also need some way to collect the names of files to be transferred. On the PC we will need to get all names of files in a given directory on disk, and on the mainframe we will need to get the names of all members of a partitioned data set.

The obvious answer for the PC side would be to issue the OS/2 DIR command for the directory of interest and parsing the returned output. However, there is a simpler way using the SysFileTree function provided with OS/2, which is explained later.

On the mainframe, the TSO LISTDS command with the MEMBERS option will display a list of all members for a given PDS. Its syntax is:

LISTDS dsname MEMBERS

A typical response might be:

```
MYUSRID.LIB.ASM
--RECFM=LRECL-BLKSIZE=DSORG
FB 80 80 PS
--VOLUMES--
D95LIB
--MEMBERS--
ABC
DEF
XYZ
```

DESIGN CONSIDERATIONS

Our preliminary design will call for the following subroutines:

- A GetFileNames routine, which will build a list of file names found in a PC directory using the SysFileTree() function
- A GetMemberNames routine, which will

hllapi('Connect',session)

Establishes a connection between your REXX program and the 3270 session specified by the single session letter.

hllapi('Disconnect',session)

Breaks the connection previously established with the Connect verb.

hllapi('SetSessionParms','options')

Changes some user-settable session parameters. This application uses the TWAIT option to cause Wait verbs to timeout after a long wait.

hllapi('QuerySessionStatus',session)

Returns information about the session. This application needs to know the number of rows and columns used for the presentation space.

hllapi('Sendkeys','string')

Simulates entry of keystrokes on the 3270 session. Each character in the string is sent as a keystroke. Use special mnemonics for special keys: @C - Clear key, @E - Enter key.

hllapi('Wait')

Waits until the 3270 session's keyboard is unlocked, that is, when it's O.K. to send keystrokes.

hllapi('Pause',halfsecs[,session])

Waits specified number of half-seconds.

hllapi('Search_PS','string')

Searches the PS buffer for a specified string and returns the buffer position of the found string.

hllapi('Copy_PS_to_str',pos,length)

Returns a string that contains a copy of the entire PS buffer.

hllapi('SendFile','pcfile session: pdsname options')

Issues a SEND command to transfer a file from the PC to the mainframe.

hllapi('ReceiveFile','pdsname session: pcfile options')

Issues a RECEIVE command to transfer a file from the mainframe to the PC.

Table 3. Commonly used EHLLAPI verbs and operands.


```

/*****
* Issue the LISTDS command for the member names
*****/
say "Getting member names for "pdsname" - Please wait..."
rc=callHLLAPI('SendKey',"LISTDS "pdsname" MEMBER @E")
rc=callHLLAPI('Pause',4,sessionID) /* give keyboard chance to lock=
*/
rc=callHLLAPI('Wait') /* wait for keyboard to unlock */
/*****
* Check that LISTDS did find a data set
*****/
rc=callHLLAPI("Search_Ps","--VOLUMES--",1) /* look for good response */
if rc=0 then
do
say "Unable to do LISTDS for" pdsname
say callHLLAPI("Copy_PS_to_str",cols+1,cols)
exit 3
end
rc=callHLLAPI("Search_Ps","--MEMBERS--",1) /* makes sure we have a PDS */
if rc=0 then
do
say "Unable to do LISTDS for" pdsname "members."
say callHLLAPI("Copy_PS_to_str",cols+1,cols)
exit 3
end
/*****
* Build the list of members while scrolling thru LISTDS output
*****/
members.0 = 0 /* no members found yet */
startlist = 0 /* clear flag to indicate we
found --MEMBERS--- header */
rc=0 /* kick start outer loop */

do while rc = 0 /* outer loop for each screen */
ps=callHLLAPI('Copy_Ps_to_str',1, rows*cols) /* get screen data */
if ps="" then /* error with hllapi call */
rc=1 /* force outer loop to quit */
else
do
do l=0 to rows - 2 /* inner loop for each line */
line = substr(ps,l*cols+1,cols) /* get a line from PS */
if substr(line,2,5) = "READY" then /* if it's READY we're done */
do
rc=1
leave /* leave innerloop */
end
member=word( line , 1) /* pick out the member name */
if member<>"*" & member<>" " & startlist & match(member, memSpec) then
do /* member name good and matched */
m=members.0 /* current number of members */
m=m+1; members.0=m; members.m = member
end
if member = "--MEMBERS--" then /* found --MEMBERS-- header */
startlist = 1 /* set flag to say so */
end
end
rc=callHLLAPI('Search_Ps',"***",(rows-1)*cols) /* look for *** */
if rc>0 then
do
rc=callHLLAPI('SendKey',"@E") /* Press Enter key */
rc=callHLLAPI('Wait') /* Wait for next screen */
end
else /* no *** on last line */
rc = 1 /* set RC to leave loop */
end
end
RETURN 0

```

Figure 2. GetMemberNames routine.

build a list of PDS member names by issuing a LISTDS TSO command.

- A routine that will issue a SEND command for each filename in a list or a RECEIVE command for each member name in a list.
- A routine to create a new PDS if UPPDS detects that the target PDS name does not exist.
- INIT routine to initialize the application. This breaks the big job into five smaller jobs.

CALLING EHLLAPI

Only one function in SAAHLLAPI.DLL provides all EHLLAPI support under OS/2. Its calling syntax is:

```
result = HLLAPI('verb' [, operands])
```

where 'verb' is a text string that identifies one of over 50 actions and operands are specific to the verb specified. Fortunately, we need only 10 of the most often used verbs for our application, which is shown in Table 3.

Before you can call an external function under OS/2 REXX, you must first register it by calling the RxFuncAdd() function. This needs to be done only once per OS/2 boot-up. Once registered, a function is known to all REXX programs:

```
RxFuncAdd('hllapi','saahlapi','HLAPISRV')
```

The result of this function is either 1 if successful or 0 if not. The first argument is the name you will use in your programs for this function. The second is the name of the OS/2 DLL in which the function resides. The third argument identifies the name under which the function is known in the DLL. Usually the first and third arguments specify the same name to avoid confusion. However, if you have a favorite name for a function or the same name is used in multiple DLLs, renaming the function at registration time works great.

INITIALIZATION

The initialization routine registers the EHLLAPI external function and prepares the user-specified session (or session A by default) as follows:

1. Connect REXX program to session (Connect verb).
2. Wait for session keyboard to unlock (Wait verb).
3. Send the Clear key to clear the screen (SendKeys verb). This is done so we do not interpret old data on the screen.

4. Wait again for the keyboard to unlock.
 5. Get the screen size (rows and columns) for later use (`QuerySessionStatus` verb).
- The session is now ready for use.

GETTING PC FILE NAMES

To upload all files from a PC directory, we need to build a list of filenames. OS/2 REXX provides a `SysFileTree()` function that will place all filenames in a user-specified stem variable:

```
SysFileTree("C:\ASM\*.*", "members.", F)
```

After the above call, the compound symbol `members.0` will contain the number of filenames found and the compound symbols `members.1`, `members.2`, up to `members.n` (where `n = members.0`) will contain the fully qualified file names, including the full path name of the directory.

GETTING PDS MEMBER NAMES

The most complex part of this application is the building of the list of PDS member names when downloading a PDS. This is where we make the most heavy usage of EHLLAPI and need to take care of some ugly details such as scrolling the output of the

LISTDS TSO command if it is longer than one screen's worth.

In TSO READY mode, when a screen's worth of data has been displayed, three asterisks are placed at the end of the output to indicate that more output is to follow. The user can hit the Enter key to get the next screen's worth of data.

Figure 2 shows the logic to do the scrolling and scanning. It consists of a double `do` loop construct. The inner loop processes each detail line on each screen of data (adding member names it scans to the list being built). While scanning, member names are checked against any wild card member name specification the user may have entered on the DOWNPDS command.

The outer loop keeps checking if the last line ended with three asterisks and, if so, sends the Enter key press. If no asterisks are found, the output from LISTDS is presumed to be complete.

TRANSFERRING THE FILES

The final step, transferring the files, consists of a small `do` loop with an EHLLAPI call for the `SendFile` or `ReceiveFile` verb (see Table 3 for syntax). Note that these verbs call the SEND and RECEIVE commands, which

will not have access to the 3270 session unless our program disconnects from the session first. Also, the return code from each file transfer should be checked. If not O.K., all further transfers are skipped (to avoid repeating the same error).

CREATING A NEW PDS

One final snag needs to be overcome: the SEND command will fail if the target PDS does not exist on the mainframe (actually, RECEIVE also fails if the target directory does not exist). To automate creation of a new PDS, the UPPDS command includes logic to prompt the user for allocation information needed to issue the ALLOCATE TSO command to create a new PDS.

PUTTING IT ALL TOGETHER

The DOWNPDS and UPPDS programs are around 320 lines each, including comments. While not perfect (for example, asynchronous messages from the operator to the TSO terminal can wreak havoc with displayed output on the screen), they exemplify many of the techniques needed to successfully use EHLLAPI.

While LU 6.2, APPC and APPN peer-to-peer communications allow tighter integration between PC and mainframe, EHLLAPI and REXX provide a powerful combination to add some new life to old applications using 3270 terminal emulation. REXX's strength in string manipulation for parsing returned output as well as building command lines make for quick, flexible, and powerful programming.

REFERENCES

The entire source code for the UPPDS and DOWNPDS programs is available in machine-readable form at the following locations:

The REXX-R-U's BBS at (303) 440-1351 in file PDSXFER.ZIP in the Info-REXX multimedia help file available from Pelt Industries.

The OS2DF2 CompuServe forum in file PDSXFER.ZIP

VTAM Programming (IBM order number SC23-0115)

IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide (IBM order number GA23-0061)

EHLLAPI Programming Reference (IBM order number S04G-1027)

OS/2 REXX online help

OS/2 Communications Manager/2 online help

Markus Pelt-Layman has been a software developer for over 22 years. He is author of several commercial software packages on IBM mainframes and PCs, including "AF/Operator" (now sold by Candle Corp), and is coauthor of the REXX interpreter in the OPS/MVS product (now sold by Legent Corp). His areas of interest include computer operations automation, computer language processing, and computer/human interface optimization. He has a B.S. degree in mathematics, and a M.S. degree in operations research from Stanford University. He is currently working on RexxAnne, a REXX compiler for OS/2, Windows NT, and Windows. He can be reached at Pelt Industries (800) 741-4322, by e-mail at MarkP@PeltInd.com, or by fax at (303) 442-3198.

Introducing...

The Software Development/Computer Language CD

Featuring:
Product Information Links with
The Programmer's Paradise® Catalog

Before you buy, get product information—fast—from the most trusted sources in the industry.

And become a more productive programmer, too! Imagine 5 years of back issues of *Software Development* and *Computer Language* magazines—available on one new CD for the first time ever. Includes feature articles, regular columns, source code, tables, and comprehensive, comparative product reviews from the June 1989 through the December 1994 issues.

The Software Development CD is the most authoritative source of practical technical information in the business.

Software Development explores the products, practices, and technologies that address issues concerning today's corporate developer. Learn from programming greats like Grady Booch, Stan Kelly-Bootle, Ed Yourdon, Warren Keuffel, Roland Racko, and Bruce Schneier about design, team

programming, languages, object-orientation, client/server, database programming, and much more.

Reliable, thorough information on programming products—fast and easy.

The Programmer's Paradise Catalog's best selling products are featured via hundreds of product highlights, informative white papers, and revealing demos electronically linked to the over 1000 *Computer Language*/*Software Development* product reviews and mentions. The industry's leading products and software publishers are represented, including Microsoft, IBM, Borland, Sheridan, ProtoView Development, Mortice Kern Systems, Lifeboat Publishing, Blue Sky Software, Apex Software, Crescent Software, LEAD Technologies, and Starbase Corporation. The easy-to-use hypertext interface lets you find the information you need—quickly!



Special Features

- Includes product demos and white papers
- Over 20,000 hypertext links—you can easily reference articles, figures, listings and more
- Full source code listings for magazine articles—use them in your own projects
- Full text searching—search for any word or phrase anywhere in any article
- Convenient bookmarks let you save your place
- Runs under Windows—use it alongside your compiler/development environment
- Look up articles by year, month, and title—or do a search

Software Development/Computer Language CD

E-MAIL ORDERS:
pkcpele@mfi.com

The *Software Development/Computer Language* CD is available approximately April 17, 1995. My price is \$49.95 plus \$3.00 shipping US/Canada (\$12.50 all other countries). California residents, add \$4.12 sales tax. New York and New Jersey residents, add applicable sales tax. Clip out or photocopy this form.

Name (please print clearly) _____ Phone Number (include area code and country code if applicable) _____

Address _____

City/State/Zip _____

Country _____

Here's how I'm paying (circle one): ☐ VISA ☐ MC ☐ AMEX ☐ Check enclosed

Credit card number (include all digits) _____

Expiration date _____

Signature required _____

OS/2 D

TO ORDER CALL:
1-800-892-1186

FAX ORDERS:
1-908-389-1390
Use this form

MAIL ORDERS:
Software Development/Computer Language CD/Attr:
Philip Keppeler 600 Harrison St., San Francisco, CA 94107

INTERNATIONAL:
Mail/fax in this order form

ONLY
\$49.95!



If you've never seen Object REXX in action, here's your chance. This product is a major improvement in the language, compatible with OS/2's SOM and Workplace Shell.

By **BRET CURRAN**

Object REXX~ HowCool?

The best way to start any computer-related article, especially if your audience consists of the guru application developers of the industry, is to jump right in with a sample program. Figure 1 does exactly that by showing you what an Object REXX program looks like.

Even though space does not allow a more descriptive analysis of what the WayCOOL program is doing (other than what the comments provide), I wanted to start by giving you a flavor of Object REXX. It should look familiar to you if you've ever seen REXX code or have done any object-oriented programming.

This article will cover the following:

- What is Object REXX?
- Using Object REXX to Work with the Workplace Shell (WPS)
- Other COOL things to do with Object REXX and WPS.

WHAT IS OBJECT REXX?

Object REXX is the follow-up version of REXX in the OS/2 world. Object REXX is currently in beta and is available via the IBM Developers Connection program (beginning with DevCon 6). There also happens to be an Object REXX for Windows, which is also in beta.

Object REXX supports both procedural and object-oriented programming, in the same program if you'd like, and will be completely upward compatible with the existing REXX on OS/2. This compatibility means that existing REXX programs, REXX function packages, and REXX products and environments will all work unchanged on Object REXX.

Who Will Be Interested in Object REXX?

Probably everyone will be interested in Object REXX, especially when it becomes an integral part of the operating system. Following are some people who will definitely be interested:

- Newcomers to object-oriented program-

ming will find that Object REXX is an easy way to get their feet wet. Object-oriented concepts can be exercised quickly and easily with Object REXX, especially compared to other tools available today.

- Object-oriented programmers will enjoy Object REXX because of its ease of use and implementation. It could easily become the object-oriented prototyping tool of choice.
- Experienced object-oriented programmers will find that Object REXX is a powerful, state-of-the-art object-oriented programming language, featuring multiple inheritance, object-based concurrency, and both class- and instance-based programming.
- WPS developers, those interested in working with the WPS, or those learning how to program the WPS interface will enjoy the capability that Object REXX provides to easily and quickly experiment with the shell's API.

```

/*****
/* Program name:  OOREXX.CMD
instance=.OOREXX`new      /* Create a new instance of the
                           /* OOREXX class defined below
instance`HowCool          /* Invoke the HowCOOL Method
/* End of executable code; start of class/method definition*/
::CLASS OOREXX            /* Define the OOREXX Class
::METHOD HowCOOL          /* Define the HowCool Method
    say ('OOREXX is WAY Cool!']]'OA'x)`copies(5)
                           /* Use the copies method (from the
                           /* STRING class) to create output
/*****

Output from WayCOOL.CMD
/*****
OOREXX is WAY Cool!
OOREXX is WAY Cool!
OOREXX is WAY Cool!
OOREXX is WAY Cool!
OOREXX is WAY Cool!
/*****
  
```

Figure 1. OOREXX: an Object REXX sample program.


```

/* Program Name:          SETOBJID.CMD          */
/* OS/2 Developer Magazine: July/August, 00 or REXX, p. ?? */
/* Article Title:         ObjectREXX WayCOOL    */
/* Author:                Bret Curran           */
/*                        Infrastructure Incorporated */
/*                        Internet: (bretc@ibm.net) */
/*                        Phone: 800-496-3042     */
/* Description:           */
/* This program will set an objectid for a specified object. */
/* You can specify the object with the WPS title and folder */
/* location. Once the ObjectID is set, then the object can  */
/* be manipulated via the ObjectID from Classic REXX        */
/* Program requirements */
/* Object REXX must be loaded */
/* Object REXX wpfind.cmd must be PATHed */

use arg argStr
args = argStr~strip
parse var args path ',' title ',' objectid

/* First, look in the 3rd parameter for an objectid */
/* If it's there, then parse accordingly, */
/* else */
/* if the objectid is in the second parameter then */
/* parse with only two parameters passed */
/* else there's an error in the parameters */
if objectid~pos('<') \= 0 then
do
    path = path~strip('B','"')
    title = title~strip('B','"')
    objectid = objectid~strip('B','"')
end
else
do
    if title~pos('<') \= 0 then
    do
        objectid = title~strip('B','"')
        title = path~strip('B','"')
        path=""
    end
    else
    do
        say 'Syntax:'
        say '  setobjid path title objectid'
        say '  where'
        say '    path is the title of folder(s) delimited by ;'
        say '    omit if the desktop is the only folder'
        say '    title is the title of the target object;'
        say '    use lf for split titles'
        say '    objectid is the desired objectid'
        say '    (must specify "<" and ">")'
        say '    all values are case sensitive'
        say '    separate the 3 parameters with commas'
        say '    examples'
        say '    setobjid "OS/2 System;Information", "Multimedia", "<MM>"'
        say '    setobjid "Internet", "<IFolder>"'
        exit 1
    end
end

/* if path is blank, then there's only two parms */
/* otherwise, concatenate the path and title with a semicolon */
/* and then have wpfind go get us the object */
if path="" then
obj=wpfind(title)
else
obj=wpfind(path';'title)

```

Figure 2. SETOBJID program (continued on page 31).

The Power of Object REXX. The power of REXX, especially on the OS/2 platform, has been increasing steadily in recent years. This is due mostly to the increased number of available development tools and libraries. REXX itself also has been improved—by at least several orders of magnitude. The fact that it is an object-oriented language brings it into another class (pun intended) of programming languages. Couple that with REXX's upward compatibility, and you have all this great new stuff while not losing anything from before.

The power of REXX extends into many different facets of programming, especially on the OS/2 platform. My personal favorite is its ability to work with OS/2's user interface, the Workplace Shell. ("Using REXX to Customize the WPS, Parts 1 and 2," *IBM Personal Systems Technical Solutions*, April, July, 1993).

Object REXX extends the power to work with the WPS much further than before, primarily because it can work with and manipulate System Object Model (SOM) objects. This means you can use and reuse SOM objects whether they were created in C, C++, or any other SOM-supported language. And because the WPS is based on SOM objects, Object REXX allows you to work directly with any of the WPS objects via the actual object methods.

USING OBJECT REXX TO WORK WITH THE WPS

Working with the WPS using "Classic" REXX turned out to be more an art form than anything. The basic functionality was provided through a few functions in REXXUTIL.DLL. While these functions allowed you to create many objects and even modify some, there were quite a few serious limitations to using Classic REXX to work with the WPS.

The fundamental limitation was that each and every WPS function or method that was added or modified had to be replicated by someone in REXXland, which caused delays and limitations in the capabilities provided with REXX. An additional disadvantage was that only object settings that had setup strings vs. just object styles could be manipulated via REXX. This meant that if you could work with a WPS object, chances are good that you could perform only a limited number of actions. And last, but certainly not least, the most frus-

trating limitation had to be WPS objects that did not have ObjectIDs. If an object did not have an ObjectID or was not a WPFileSystem object, there was just nothing you could do with the object from a REXX perspective.

Object REXX to the Rescue. All of those limitations are history when you move to Object REXX. Essentially, you can work with any WPS object just as if you were programming from a C or C++ perspective (only more easily). You can run any WPS method on the object, compared with a predetermined finite set of REXX functions. Plus, the ObjectID limitation is gone as well, although you can still use it if you'd like. With these limitations removed, you can see that the world of Object REXX and WPS is going to be quite an interesting one, and the easiest way to see that is with an example.

The SETOBJID Example. SETOBJID (Figure 2) is an example of an Object REXX program that provides a solution to a problem with Classic REXX. Remember, Classic REXX can work with an object only if it has an ObjectID. The SETOBJID program will allow you to specify an object by

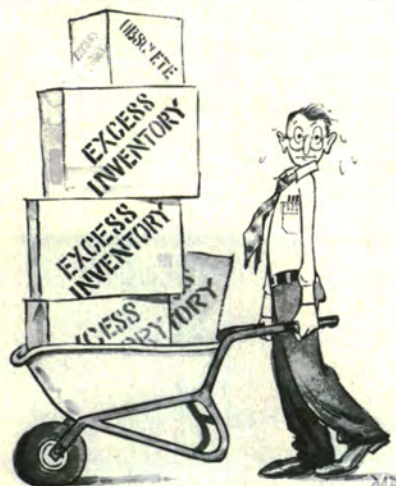
```
/* if wfind returns a number, then it failed for our purposes */
if datatype(obj) = 'NUM' then
do
  say 'Bad return code from wfind call -- 'obj
  exit 2
end

/* find current ObjectID of object, if it exists */
currObjid=obj~wQueryObjectId

/* if none exists, then we'll get 'The NIL Object' returned */
/* Test to make sure objectid should be overwritten */
if currObjid \= 'The NIL Object' then
do
  say
  say 'The 'title' object currently has an ObjectID of 'currObjid
  say 'Are you sure you want to replace it with 'objectid'? <y/n>'
  say
  pull answer
  if answer \= 'Y' then
    exit
end

/* Now that everything is in order, let's make the simple call */
/* to actually set the objectid for the object */
obj~wSetup('OBJECTID='objectid)
say
say 'The 'title' object now has an ObjectID of 'objectid
exit
```

Figure 2. SETOBJID program (continued from page 30).



Need some help with that?

Turn your excess inventory into a tax break and help send needy kids to college.

Call for your free guide to learn how donating your slow moving inventory can mean a generous TAX WRITE OFF for your company.

Call (708) 690-0010

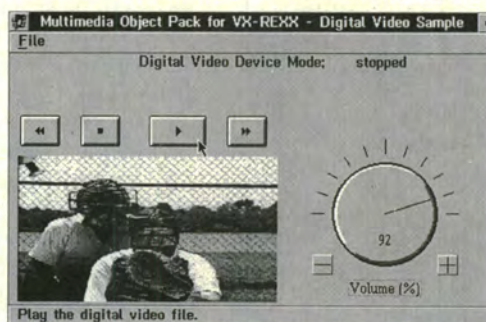


P. O. Box 3021 • Glen Ellyn, IL 60138 • Fax (708) 690-0565

Excess inventory today....student opportunity tomorrow

Multimedia Object Pack for VX-REXX

New! Version 1.0.1



\$99.95
Includes royalty-free distribution rights.

Develop professional quality multimedia applications with VX-REXX!

- Circular Slider and Animated Graphic Push Button objects.
 - Create custom animation frame sequences!
 - Real-time control of OS/2 multimedia devices.
 - ampmix, cdaudio, cdxa, digitalvideo, sequencer, waveaudio, and videodisc.
 - Notify, NotReady, Pause, Play, Record, Seek, and Stop notification events.
 - Play digitalvideo movies in an Ultimotion window or any VX-REXX window.
- You can even play movies and animations in a VX-REXX entry field object!

Get started developing state-of-the-art multimedia applications! Order now and receive the Multimedia Object Pack for only \$99.95 (normally \$150).

DB Technologies, Inc.

2420 Briar Oak Circle
Sarasota, Florida 34232-6100

Sales Department: 800-830-8703

Tech Support: 813-378-3760

Fax: 813-377-7392

CompuServe: 72123,3661

Internet: 72123.3661@compuserve.com

VX-REXX is a trademark of Watcom International Corporation.
OS/2 and Ultimotion are trademarks of the IBM Corporation.



REXX Diagnostic Commander Makes REXX Debugging Easy

Software developers, administrators, and all OS/2 professionals can efficiently code and test REXX command files in a completely interactive environment. Easy to use, RDC enhances your code writing ability while increasing productivity. *Programs are debugged easily and quickly!*

- Supports Multiple Platforms
- Source Code Display
- Variable Display Service
- Variable Alteration
- Single Step Review
- Dynamic Watch Windows
- Logic Alteration
- Deferred Debug Option

Get the REXX Diagnostic Commander

Only \$45.00

Order It Today!

**Call
800-724-7011**



**Suitable
Alternatives**

4655 Old Ironsides Drive, Suite 220
Santa Clara, CA 95054 • 408-727-3142



its title and folder and then set an ObjectID for that specific object. With the ObjectID set, even Classic REXX can manipulate the object.

The most complex part of the program is the actual parsing of the parameters for the different possible scenarios. Some of the complexity is due to the way `wpfind.cmd` looks for its parameters. As you can see from the code, if an object is nested within several folders, `wpfind` wants a path of titles for those folders starting with the first folder on the desktop. One possible change to the `'wpfind.cmd'` is to make it accept parameters in a little more straightforward fashion; it would first ask for the title of the object and then for the folder where the object is.

Note that I've used methods wherever I could, even in the parsing and verification of the parameters. Object REXX allows you to use either the built-in functions you are used to or the built-in methods.

Once the parameters are parsed, the rest of the program is fairly easy. Before setting the ObjectID, the `wpQueryObjectId` method is called. This is to get a confirmation from the user that if a current ObjectID already exists, it should be overwritten.

One interesting thing with the SETOBJID program is that it could be written, or at least tested, in just two lines! I first experimented with the capability with two commands using REXXTRY:

```
obj=wpfind("Internet")
obj~wpsetup("OBJECTID=?TestObjectId>")
```

One other note is that multiline titles will cause problems. This is because there is a linefeed character (hex '0A') in the title. I've left it as an exercise for the reader to add this logic (which really means I'm running out of time and space).

OTHER COOL THINGS TO DO BBWITH OBJREXX AND WPS

The surface of what Object REXX can do has barely been scratched, even when talking about just the limited scope of using Object REXX to work with the WPS! One cool thing to try when you get your version of Object REXX going is to start up a session of REXXTRY. Then get out your WPS reference documentation, which comes on the Developer's Connection as INF files (WPS1+WPS2+WPS3). Spin through some different methods, such as `wpQueryIconTitle`, `wpOpen`, and

`wpQueryObjectId` and see how easy it is to manipulate WPS objects—there's no quicker way to learn the WPS.

Another huge topic that has not even been touched on is the ability of Object REXX to actually subclass SOM and WPS objects. Yes, it will work. You can create REXX classes that are subclassed from SOM WPS classes. Once that is done, you can override the inherited methods to make the WPS object behave the way you want it to. The traditional example (funny how "traditional" in this industry is one- or two-year-old techniques) is to create a password-protected folder by subclassing `WPFolder` and overriding `wpOpen` with the logic to open the folder only after a valid password has been entered. The Object REXX developers are providing an example that will be doing just that with the Object REXX code on the Developers Connection 7 CD-ROM.

Bret Curran is an independent OS/2 systems consultant, specializing in assisting corporations install, configure, and manage large OS/2 networks. He was previously the OS/2 team leader in IBM's Personal Systems Competency Center in Dallas, Tex. He has a bachelor's and master's degree in Information Systems from the University of North Texas. You can reach Bret at curranb@ibm.net or 817/382-2362.

REFERENCES

Object REXX Reference (OBJREXXO.INF) in Object REXX WPS References (WPS*.INF) in OS/2 Toolkit

The Object REXX code can be found on the Developers Connection CD-ROM, beginning with DevCon 6. This article and sample code are available on CompuServe's OS2DF2 Forum, OS/2 Developer section. Download file XXXXXX.ZIP.

Thanks to the entire Object REXX development team for creating such a cool and powerful language!

Thanks especially to Steve Pritko and Rick McGuire for the help he provided, both knowingly and unknowingly.



A well-designed program is the most effective way to gain performance improvements. But sometimes code tuning is the only practical way to do it. These tips and techniques should help.

By **CRAIG HODGINS**

Six Simple Code-Tuning Techniques

Code tuning is not necessarily the best method to improve program performance. In fact, it is often viewed as a very negative thing to inflict upon any program. For example, in Steve McConnell's *Code Complete: A Practical Handbook of Software Construction*, Donald Knuth says, "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."

A well-designed program is a more effective way to gain performance improvements. Good data structure and algorithm selections also bring better results. Buying new hardware or a better compiler is an easier way to improve performance. But sometimes code tuning is the only practical way to improve substandard performance, especially when using REXX, which is for the most part an interpreted language.

The Pareto Principle, commonly known as the 80/20 rule, applies to program optimization, as well as other areas of life. This

principle states that you can get 80% of the result with only 20% of the effort. As documented in McConnell's handbook, Barry Boehm found that 20% of a program's routines consume 80% of its execution time. Knuth discovered that less than 4% of a program usually accounts for more than 50% of its run time.

What this means is that very often small snippets of code, often overlooked by programmers, can be costly in performance. Programmers must identify those small parts of the program that are causing the performance degradation. The performance can be measured, the hot spots identified and optimized, and the performance measured again to see if an improvement occurs.

The measurements must be precise and consistently applied. A simple way to do this in REXX is to use the language itself. See "A Simple Performance Analyzer" for a discus-

```
flag = 1
sum = 0
product = 0
num = 100000
x = time('E')
do i = 1 to num
  if flag = 1
    then
      do
        sum = sum + 1
      end
    else
      do
        product = product * i
      end
    end
  end
say **** Elapsed time: 'time('E')
```

Figure 1A. Unswitching.

```
flag = 1
sum = 0
product = 0
num = 100000
x = time('E')
if flag = 1
  then
    do
      do i = 1 to num
        sum = sum + 1
      end
    end
  else
    do
      do i = 1 to num
        product = product * i
      end
    end
  end
say **** Elapsed time: 'time('E')
```

Figure 1B. Unswitching.

sion of a simple performance analyzer tool.

Once a hot spot has been identified, how do the programmers go about improving performance? Many code-tuning techniques are based on established programming principles.

What follows are six of the most common and simplest techniques and how they can be applied while using the REXX language.

Please note that the code examples shown are for illustration purposes only and do not necessarily represent

practical logic constructs. In fact, some of them may appear trivial. However, they do illustrate the code-tuning principles under discussion. The loop counters may also seem exaggerated in some instances, but remember that in a real application, thousands of iterations may be involved, and the point is to show how a small change in the code can affect elapsed time.

SIX CODE-TUNING TECHNIQUES

Unswitching. Unswitching turns a loop inside out. A switched loop makes a decision every time it is executed (Figure 1A). Since the decision doesn't change as a result of the looping going on around it, you can unswitch the loop by moving the decision outside the loop. Basically, unswitching is putting loops inside the conditional rather than putting the conditional inside the loop.

In Figure 1A, the loop is executed *Num* times. The *Flag* variable is tested inside the loop, and different branches are taken accordingly. In Figure 1B, the *Flag* variable is tested outside the loop, and the appropriate loop is executed as a result of the test. By moving the test outside the loop, a performance improvement of 33% occurs.

Jamming. Also known as fusion, this technique combines two loops that operate on the same set of elements. Figure 2A shows two loops that are candidates for jamming because they are both executed *Num* times and contain arrays of the same number of elements.

Figure 2B shows the jammed (or fused) loop. The performance gain comes from cutting loop overhead; you have exactly one-half the number of loops as before, with the same resulting function. Note that the performance is not cut in half, as might be expected. This shows the importance of actually measuring performance results, rather than assuming what might happen.

Unrolling. Similar to jamming, unrolling reduces loop housekeeping, but in this instance by handling two or more cases in each pass through the loop. Figure 3A shows a While loop that can be unrolled. In Figure 3B, the loop has been unrolled once; that is, two assignments are done in the loop and the loop counter is incremented by two, instead of one. The loop could be unrolled three times, four times, or more.

Be warned, however, that unrolling costs heavily in readability and maintainability. You can see that un-

```
num = 10000
x = time('E')
do i = 1 to num
  CustomerName.i = ' '
end
do i = 1 to num
  balance.i = 0
end
say '*** Elapsed time: ' time('E')
```

Figure 2A. Jamming.

```
num = 10000
x = time('E')
do i = 1 to num
  CustomerName.i = ' '
  balance.i = 0
end
say '*** Elapsed time: ' time('E')
```

Figure 2B. Jamming.

```
num = 20000
i = 1
x = time('E')
do while (i <= num)
  anyvar.i = i
  i = i + 1
end
say '*** Elapsed time: ' time('E')
```

Figure 3A. Unrolling.

```
num = 20000
index = 0
i = 1
x = time('E')
do while (i < num)
  anyvar.i = i
  index = i + 1
  anyvar.index = i + 1
  i = i + 2
end
if i = num then anyvar.num = num
say '*** Elapsed time: ' time('E')
```

Figure 3B. Unrolling.

A Simple Performance Analyzer

You don't have to spend a lot of money to get a good performance analyzer. REXX comes with a *TIME* function that can be effectively adapted to provide a simple yet useful performance tool. The benchmarks used in this article were all calculated using this technique.

The *TIME* function by default returns the local time. Various options may be used to obtain alternative formats or gain access to the elapsed time clock. *TIME('R')* returns the number of seconds and microseconds since the elapsed time clock was started or reset. It also resets the elapsed time clock to zero. *TIME('E')* returns the number of seconds and microseconds since the elapsed time clock was started or reset.

The *TIME* function can be used for measuring real elapsed time intervals. The first call to either *TIME('R')* or *TIME('E')* will return 0. Thereafter, the elapsed time since the first call or since the last call to *TIME('R')* will be returned.

To find the hot spots in your REXX program, insert *TIME* calls at strategic places throughout the code, making sure to label them appropriately so you can match the times with what is being measured. In this way, you have a reliable, consistent measurement with which to make your benchmarks. The benchmarks for this article were evaluated by inserting *TIME('E')* statements at the beginning and end of the code to be analyzed, along with a *SAY* statement to write the elapsed time to the screen.

For the most accurate results, do the benchmarking in isolation from any other computer functions. In other words, do not have any other system tasks or programs running while you are analyzing the performance of your REXX program. Other tasks and programs may take up clock time and corrupt your benchmark results.

rolling the loop just once adds more lines of code and more logic than the original. Also, code must be added to cover special cases, since the counter is incremented by two instead of one.

Sentinel Values. In this article, a sentinel value is a variation of the classical sentinel value concept. In the classical case, a loop has a compound test; for example, a search loop that tests whether a match has been found and also whether or not the search list has been exhausted. The code tuning is achieved by inserting a test value at the end of the search list and then changing the loop conditional to test whether or not the current search element equals the test value.

While the conditional is not equal, the loop increments; when it is equal, the loop is terminated. Because of this earlier termination, processing time is reduced.

The variant sentinel value discussed here is based on a similar idea. In our case, a test value is assigned, and the conditional checks whether the current array element is equal to the test value. If not, the loop is incremented; if so, the loop terminates immediately, because the matching value has been found.

Other code goes with the sentinel value sample. For example, for Figure 4A an If Found = Yes statement group is needed to determine whether the test value actually was found. In Figure 4B, the code would be similar to an If ($i \leq \text{maxcount}$) group.

For our example, the test value variable is arbitrarily set at 7777 in an array of 10000 for consistency. As the test value changes, the elapsed time would change also; however, the time savings would still be achieved by using the sentinel value.

Busy Loops Inside. When loops are nested, such as in Figure 5A, care must be taken as to which loop to place on the inside. This loop loads a two-dimensional table with random numbers. The table has five rows and 100 columns. With the busiest loop on the outside, the performance is not bad.

However, when the busiest loop is placed on the inside, the performance improves 20%. On every iteration, a loop has to initialize the index, increment the counter, and check it. For Figure 5A, the total number of executions is 100 for the outer loop and 500 for the inner loop (100 x 5). This is a total of 600 iterations.

Switching the loops changes the number of outer iterations to five and the number of inner iterations to 500

```
maxcount = 10000
do i = 1 to maxcount
  list.i = i
end
found = 0
i = 1
testvalue = 7777
x = time('E')
do while ((found = 0) & (i < maxcount))
  if list.i = testvalue
    then
      do
        found = 1
        say '*** Found ' list.i
      end
    else
      do
        i = i + 1
      end
    end
  end
end
say '*** Elapsed time: ' time('E')
```

Figure 4A. Sentinel value.

```
maxcount = 10000
do i = 1 to maxcount
  list.i = i
end
i = 1
testvalue = 7777
x = time('E')
do while (list.i <> testvalue)
  i = i + 1
end
say '*** Elapsed time: ' time('E')
```

Figure 4B. Sentinel value.

```
maxrows = 5
maxcols = 100
x = time('E')
do c = 1 to maxcols /* load
  random table by row, column */
  do r = 1 to maxrows
    table.r.c = random()
  end
end
say '*** Elapsed time: ' time('E')
```

Figure 5A. Busy loops inside.

```
maxrows = 5
maxcols = 100
x = time('E')
do r = 1 to maxrows /* load
  random table by column, row */
  do c = 1 to maxcols
    table.r.c = random()
  end
end
say '*** Elapsed time: ' time('E')
```

Figure 5B. Busy loops inside.

```
x = random()
y = random()
say 'x = ' x
say 'y = ' y
a = time('E')
if foobar(x) < foobar(y)
  then
    do
      say 'Foobar(X) is less than
Foobar(Y)'
    end
  else
    do
      say 'Foobar(X) is greater than or
equal to Foobar(Y)'
    end
  end
say '*** Elapsed time: ' time('E')
exit
foobar: procedure
arg n
sum = 0
/*say n*/
do i = 1 to n
  sum = sum + i
end
do j = 1 to n
  sum = sum + (n * 2)
end
say 'sum = ' sum
return sum
```

Figure 6A. Simplify tests.

```
x = random()
y = random()
say x
say y
a = time('E')
if x < y
  then
    do
      say 'X is less than Y'
    end
  else
    do
      say 'X is greater than or equal to
Y'
    end
  end
say '*** Elapsed time: ' time('E')
```

Figure 6B. Simplify tests.


```

parse upper arg debug_flag

if debug_flag = '/Y'
then
do
!BD! = ''          /* debug is on */
!ED! = ''
end
else
do
!BD! = '/*'        /* debug is off */
!ED! = '*/'
end

do i = 1 to 5
interpret !BD! say " i= " i !ED!
/* must use double quotes
*/
interpret !BD! say i !ED!
end

interpret !BD! say " *** loop is done
*** " !ED!

exit

```

Figure 7. Dynamic SAY.

Test	Elapsed	% Saved	Iterations
1A	26.06		100000
1B	17.49	33	100000
2A	6.68		10000
2B	4.68	30	10000
3A	8.52		20000
3B	7.83	8	20000
4A	3.08		10000
4B	1.93	37	10000
5A	0.25		600
5B	0.20	20	505
6A	0.43		1
6B	0.03	93	1

Elapsed times are in seconds based on an average of three executions.

Table 1. Benchmark results.

(100 times 5). This results in 505 executions instead of 600, and the resultant performance savings.

Simplify Tests. Use simplified tests rather than more costly operations to

save on performance. For example, Figure 6A shows a program that compares FOOBAR(X) and FOOBAR(Y). The FOOBAR internal function is intended to be meaningless but time consuming. You can substitute your favorite function for FOOBAR.

Since the function acts identically on the arguments, any comparison made on the arguments will equal any comparison of the results returned by the functions. In other words, by replacing the FOOBAR comparison with a simple $X < Y$ comparison, performance is vastly improved.

Figure 6B illustrates this simple improvement. The arguments X and Y are compared, rather than the returned values from the FOOBAR functions. The results speak for themselves.

Sometimes programmers like to use the neat features of a language to keep code tight and compact, thereby sacrificing performance and, in this case, readability.

SUMMARY

Code tuning is not always the best way to improve performance. Performance should be built-in by good module design and correct data structure and algorithm selection. But when tuning is required, remember these six simple code tuning techniques and the principles on which they are based.

Craig Hodgins has worked for IBM Canada Ltd., Toronto, Ontario for 14 years in both systems and application programming. Currently, he is a programmer/analyst working on a large software manufacturing system. Craig earned an Bachelor of Arts in History from York University in Toronto in 1981. Craig can be reached at (905) 316-5858 or chodgins@vnet.ibm.com.

A Simple Debugging Tool

One of the best and simplest debugging tools available in REXX is the **SAY** statement. As the programmer, you can place **SAY** statements throughout your code to issue explicit debug messages or see exactly what is in a variable at run time.

However, in a large, complex program, many **SAY** statements may exist. Once the program is debugged and in production, the **SAY** statements are no longer needed—until a change is made and debugging is necessary again.

It can be time-consuming and frustrating to have to apply all those **SAY** statements again, especially in the right spots. The following technique can be used to dynamically turn the **SAY** statements on and off with a simple switch.

Figure 7 shows a simple use for the dynamic **SAY** technique. When the REXX program is called from an OS/2 window with the **/Y** parameter, the **SAY** statements in the program will be "turned on" and executed. If the **/Y** parameter is not passed, the **SAY** statements will be interpreted as comments and not executed.

This is accomplished by using the **INTERPRET** statement to dynamically turn the **SAY** statement on and off. The **!BD!** (begin debug) and the **!ED!** (end debug) control characters are set to either nulls or comment delimiters. When set to nulls, the **SAY** statement is executed. When set to comment delimiters, the **SAY** statement is interpreted as a comment. In this way, the **SAY** statements can be left in the program to be used or not used as required.

This technique has two drawbacks. First, you cannot use it with compiled REXX, as the compiler does not understand the **INTERPRET** statement. Second, in an interpreted environment, the **INTERPRET** statement uses resource, whether the **SAY** statements are turned on or off. This must be taken under consideration.

Like most things in computer science, a tradeoff exists between performance and ease-of-use that only the individual programmer can weigh at any given time in any given situation. Regardless of the practical use, the dynamic **SAY** statement may have in the real world, it illustrates quite nicely the flexibility and power of the REXX language. Creative programmers have no lack of opportunity with REXX.

REFERENCES

Cowlshaw, M. F. *The REXX Language: A Practical Approach to Programming*. Englewood Cliffs, N.J., Prentice Hall, 1990.

McConnell, S. *Code Complete: A Practical Handbook of Software Construction*. Redmond, Wash.: Microsoft Press, 1993.

REXX Reference (IBM S10G-6268-00).

REXX User's Guide (IBM S10G-6269-00).



REXX is an international language. Here's how to design a program that is translation-ready.

By BRUCE ERIC HOGMAN

Designing REXX Programs for Multiple Languages

Today, REXX programs can be distributed around the world via the Internet and other media. You can prepare for worldwide distribution by designing a program to facilitate its translation to other national languages such as Spanish or German. This article addresses some design considerations for translation-ready applications. OS/2 REXX is not changed for other language versions of OS/2, other than translating the error messages from the REXX interpreter. Even though REXX functions such as `STREAM()` use operands that are English phrases such as "query exists", these operands remain unchanged in other language versions of OS/2. The result codes from functions such as `STREAM()` that are English words such as "ERROR" or "READY" remain unchanged also. A REXX program written for the U.S. English version of OS/2 will run unchanged on another language version of OS/2, assuming that the operating system version is similar. For example, if you use functions unique to OS/2 Warp V3.0, your program will not run on earlier releases of the operating system that do not support the functions.

OS/2 COMMAND NAMES

Other language versions of OS/2 retain the U.S. version command names. This means you can invoke OS/2 commands by name, such as `COPY` or `RENAME`, within your REXX programs, and the program will not need to be changed to work with other language versions of OS/2.

Messages from running OS/2 commands are changed, however, so your REXX programs cannot depend on information remaining the same if you invoke a command and capture the `STDOUT` output. You should design your REXX program to interpret the return codes from invoked commands and

to do additional processing to ensure the successful processing by OS/2 commands. For example, if you copy files using the `COPY` command, successful completion results in `RC=0`. It is good practice in general to test return codes from all commands and functions. Determine what return codes are produced by all the commands your program will use by first reviewing the online OS/2 Command Reference (`VIEW CMDREF`) and then by coding and running test REXX programs to invoke the commands and display the resulting return codes.

EXTERNALIZING MESSAGES

To design for national language support (NLS), plan to package all messages from your program in external data files. The message file can be translated into other languages, with each different language identified by its one- or two-character language identifier associated with the `SYSLEVEL` of the OS/2 system as shown in Table 1.

You can save space in distributing your application files by using the `PACK.EXE` utility that is part of the Developer Toolkit to create a bundle file containing all language versions of your application files. You can invoke the `UNPACK` command that is distributed with OS/2 to extract those files specific to the language in use on the PC. Name the packed files with a common prefix and language-specific suffix, like `MYAPP.M_0`, `MYAPP.M_F`, `MYAPP.M_G`, where you will extract the file `MYAPP.M_?` to the hard drive and rename it to `MYAPP.MSG` for use by your REXX program.

If you pack files and your program will be run on early releases of OS/2, restrict yourself to the `PACK.EXE` program instead of the later `PACK2.EXE` so that the user can run `UNPACK` successfully. Refer to the Developer's Toolkit online documentation

1 OS/2 country name
 2 OS/2 keyboard name
 3 OS/2 countrycode (3 num) (CONFIG.SYS)
 4 OS/2 codepage (3+3 or 3) (CONFIG.SYS)
 5 OS/2 keyboard code (5 char) (CONFIG.SYS)
 6 OS/2 Syslevel 3rd char (1 char)

1	2	3	4	5	6
Brazil	Brazilian	055	850,437	BR	B
Canada	Canadian French	002	850,863	CF	C
(French)					
Denmark	Danish	045	850	DK	D
Finland	Finnish	358	850,437	SU	L
France	French (189)	033	850,437	FR189	F
Germany	German	049	850,437	GR	G
Italy	Italian (141)	039	850,437	IT141	I
Netherlands	Dutch	031	850,437	NL	H
Norway	Norwegian	047	850	NO	N
Portugal	Portuguese	351	850,860	PO	P
Spain	Spanish	034	850,437	SP	S
Sweden	Swedish	046	850,437	SV	W
U.K.	U.K.	044	850,437	UK166	U
U.S.	U.S.	001	437,850	US	0

Table 1. Language and country codes.

softcopy for full information on the use of the PACK utility.

Include in your program initialization code to load the message file into a stem variable at the start of your program, to permit fast access to the data. Use code similar to that included below:

```
/* load messages from external file */
/* make stem variable "message." all null */
message. = ''
msgfile = 'MYAPP.MSG'
do while lines(msgfile) > 0
  xstr = linein(msgfile)
  /* extract the message identifier */
  msgid = left(xstr,7)
  /* id and text into stem variable */
  message.msgid = xstr
end
/* close the input file */
call stream msgfile,'c','close'
```

In the above block of code, the messages are addressed using their message identifier, the three-letter prefix, and four-digit suffix. The message identifier and text for message MYA0010 is in variable message.msgid when msgid = 'MYA0010'. Display both the message identifier and the message text to the user. This helps support staff to resolve problems by identifying the message clearly. Avoid using a single message for more than one logical function. This helps to

identify the code in your program that issued the message. Some messages, such as generic data display, could be reused.

The \os2\install\syslevel.os2 file, byte 47, is the single-character language code that indicates the language version of OS/2 installed on the personal computer. The IBM external function package RXUTILS, available on IBM BBS, IBM internal VM OS2TOOLS, and other sources has a function, RXSYSLEVEL, that returns the SYSLEVEL value and associated identifying text. Review the details on that function in the RXUTILS.INF file.

The following REXX code, when created as SYSVALU.CMD file, can be used to obtain the OS/2 SYSLEVEL if you do not have RXUTILS function package:

File SYSVALU.CMD:

```
/* SYSVALU function: given filespec of syslevel.
return syslevel string */
arg slvlfile
if slvlfile = '' then return ''
sdata = charin(slvfile,,chars(slvfile))
call stream slvlfile,'c','close'
/* extract the syslevel and text */
return substr(sdata,45,pos('00'x,sdata,45))
```

In another REXX program, re-

trieve the SYSLEVEL.OS2 value by using the following code:

```
bootdrive = left(value('COMSPEC',,
  OS2ENVIRONMENT'),2)
/* invoke sysvalu.cmd as external
function (must be in PATH) */
syslevel = 'SYSVALU'(bootdrive'\os2\
  install\syslevel.os2')
lang_code = substr(syslevel,3,1)
```

FORMAT OF MESSAGE FILE

Your message file should have a format similar to that shown below:

```
XXXnnnnS TTTTT %1 TTTTT %2 TTTTTTTT
```

XXXnnnnS is a message identifier with a constant letter prefix of three characters, and a numeric suffix of four decimal digits, with a severity indicator letter from the following list:

E: Error
 H: Help
 I: Information
 P: Prompt
 W: Warning

An example line entry in your message file may appear as follows:

```
MYA0010I: File %1 created.
```

This example follows the syntax requirements of the Developer's Toolkit message file utility input text file. If you use the same message file format, then your message file could later be used if you convert your REXX program into a compiled language program. In any case, your messages would appear similar to those displayed by IBM programs, and all your messages would have a common appearance. Your REXX code can read in the message file as a text file.

The percent sign and number in the above message example indicate a text insertion into the message text, where an argument passed to your message displays procedure or function substitutes for the text insertion tag. Let's assume you name your message display procedure Display_Message;, then you would retrieve message text using code similar to that below:

```
msgtext = Display_Message('MYA0010',
  created_file)
```

You could then use the SAY keyword to display the message text or log the text to a file. The Display_Message procedure would read the message

file or table, addressing the message. stem variable similar to the code shown here:

```
Display_Message: procedure expose
    message. /* input: 7-char message
    identifier
plus optional 1-9 text arguments:
msgid,insert1,insert2,insert3, ...
returns: message text with insertions */
/* obtain message identifier argument */
parse arg msgid,other
/* make identifier upper case */
msgid = translate(msgid)
/* get text of message entry */
xstr = message.msgid
/* get position of '%' in text */
ki = pos('%',xstr)
do while ki > 0
inum = substr(xstr,ki+1,1)
if datatype(inum,'w') & ,
(inum > 0 & inum < 10) then
do
if ki > 1 then lstr = left(xstr,ki-1)
else lstr = ''
/* code assumes only 1 digit for number */
rstr = substr(xstr,ki+2)
inum = substr(xstr,ki+1,1)
/* start with arg(2), which is the first
text insertion */
xstr = lstr||arg(inum+1)||rstr
/* find next insert or end loop */
ki = pos('%',xstr)
end
else ki = 0 /* end loop */
end
return xstr
```

The procedure code would also obtain the text for message MYA0010, parse the text, and substitute the string contained in the variable created_file for the text insertion string %1. The resulting message shown to the user would be similar to:

```
MYA0010I: File c:\myapp\newtext.txt
created.
```

DESIGNING USER RESPONSES

Avoid words or letters as responses to prompts to the user for input. This pertains to responses on which your program will base further processing. You can ask the user for responses that are character strings that do not need translation, such as path or file names. Otherwise, you can display a list of choices to the user and accept a number corresponding to one of the available listed choices when the user must choose among processing alternatives.

If you display several lists of choices during your program, and

there are similar choices listed, try to keep the order of the choices the same for each displayed list. This helps to minimize user confusion when reading the list. If you provide a default choice, that choice should be the one the user would be most likely to select in normal processing. Any default choices should follow the general principle of systems design: that a default action results in the minimum change to the system data. Keep data if it existed before your program, for example.

Take some time to design response formats carefully when dealing with dates or time of day. Remember that other countries use different date and time formats. You can access the OS/2 country information using the REXXUTIL SYSINI() function call to read the PM_National application and its corresponding keys. Review the documentation on the SYSINI() function in the REXX.INF online softcopy.

The iDate key indicates the date format, and iTime key the time format. Your prompt messages should include an example of how to format the response by displaying the current date or time using the OS/2 PM_National formats chosen by the user. The code examples listed below show you how to access this data.

Obtaining the iTime format:

```
iDate = sysini('user','PM_National',
    'iDate')
/* values are: 0,1,2: m/d/y, d/m/y,
y/m/d */
```

Obtaining the sDate separator character:

```
sDate = sysini('user','PM_National',
    'sDate')
/* single character, like '-' or '/'
for m-d-y or m/d/y */
```

Obtaining the iTime format:

```
iTime = sysini('user','PM_National',
    'iTime')
/* values are: 0, 1: 12-hr, 24-hr */
```

Obtaining the sTime separator character:

```
sTime = sysini('user','PM_National',
    'sTime')
/* single character, like ':'
for hh:mm:ss */
```

The application has other keys for list separator character, decimal point

character (which is a comma in European countries), thousands separator, and so on, including the country code itself.

The following code displays the list of defined keys for PM_National:

```
/* display ini keys for PM_National */
call sysini 'user','PM_National',
'all:', 'pmkeys.'
say 'OS2.INI PM_National keys'
do j = 1 to pmkeys.0
keyvalue = sysini('user','PM_National',
    pmkeys.j)
say copies(' ',3)||pmkeys.
j|| '=' || keyvalue
end
```

ADVANCED CONCEPTS IN LANGUAGE SUPPORT

REXX programs can handle languages with double-byte character sets (DBCS) if you use the statement below to tell the REXX interpreter you will be using DBCS strings:

OPTIONS 'ETMODE' 'EXMODE'

If used, OPTIONS should be the first statement in your REXX program so the interpreter can process your program correctly.

Handling right to left (RTL) languages such as Hebrew and Arabic can be done on a version of OS/2 not equipped to process RTL if you write functions to display message text in RTL mode. Presenting non-RTL data within RTL messages requires discussion of RTL languages that is beyond the intended scope of this article.

SUMMARY

Writing REXX programs for several language versions of OS/2 is easier using the concepts presented in this article. Several of the topics apply to REXX programs in general, such as the proper use of system settings selected by the PC user. Users will accept your applications much more readily if they find that it recognizes their personal system setting preferences, and your applications may find acceptance in several languages when you build in language support.

Bruce Eric Hogman has been programming since 1965, working mostly with IBM systems. He is presently a senior systems engineer with Electronic Data Systems Corp. and works as a consultant for the IBM Personal Operating Systems division, Customer Support, Migration Support group, in Boca Raton, FL. Hogman can be reached via Internet at bruce_hogman@bocaraton.ibm.com.



Objects

The OS/2 Workplace Shell provides a 32-bit, SOM-based interface. Here are more tips and techniques for exploiting it from your REXX programs.

By **RONY G. FLATSCHER**

The Workplace Shell: Objects to the Core

The design of the OS/2 Workplace Shell (WPS) makes it possible for REXX programs to interact directly with it. The steps necessary to successfully create, change, open, and delete Workplace Shell objects with REXX are explained here using sample REXX programs. After reading this article, you should be able to write your own REXX programs for manipulating Workplace Shell objects.

The 32-bit versions of OS/2 include a user interface based on the ground-breaking System Object Model (SOM) technology: the Workplace Shell. Some simple interfaces to the WPS have been supplied for REXX via the `RexxUtil.DLL`. In order to utilize this interface and understand our examples, a basic understanding of the WPS is necessary.

The Workplace Shell is built on a SOM-based class hierarchy called "The Workplace Object Class Hierarchy" (Figure 1). The root

of all of WPS object classes in SOM is clearly visible, as the class `WPObject` is an immediate descendant of the abstract `SOMObject` class. Abstract classes are not meant for producing objects directly from them, but instead for ordering properties in a structured, hierarchical manner. Hence, abstract classes are used as a base for more refined classes and usually implement general properties found in all descendants.

Due to the object-oriented technology in WPS, every object class in its hierarchy inherits all capabilities of its ancestors, refines them, and may add additional properties. A programmer simply asks for an instance of a nonabstract class and gets a fully functional object in return, one which incorporates all the capabilities of the class and its ancestors.

SET-UP STRING INTERFACE

Some WPS classes allow for a very simple

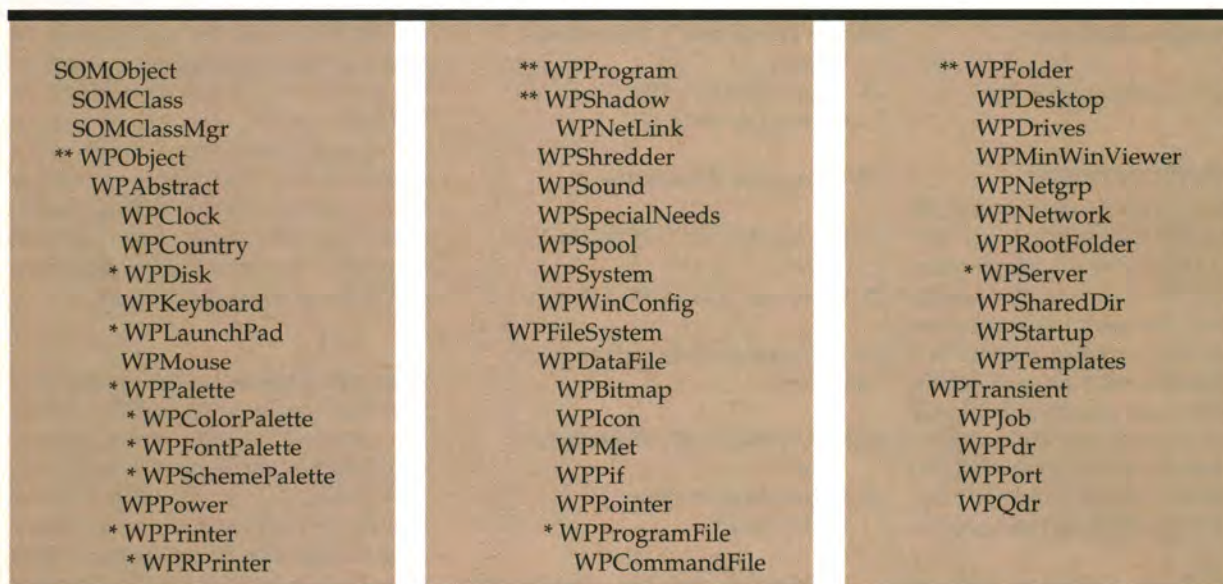


Figure 1. The workplace object class hierarchy.

(and therefore very clever) interface: the setup string interface. The string interface consists of pairs of keywords and string values delimited always with a semicolon (;). Sending such strings to objects of the WPS object class hierarchy affects the state of those objects and is thereby a valid means of communication. Because the user interface is so important to OS/2, some procedural interfaces to WPS objects have been implemented, such as `WinSetObjectData()`, to allow sending setup strings to them. Some of those Win-API interfaces are available through the `RexxUtil.DLL`.

Figure 1 depicts classes with a string interface by a leading asterisk. Because the root of the WPS-tree, the abstract class `WPObj`, contains a string interface itself, *all* WPS classes contain a string interface. So the word "some" in the very first sentence of this subsection has to be changed to "all WPS-classes...". This is an immediate consequence of the basic inheritance property of object-oriented systems. Some classes, like `WPDisk` add additional strings, that is, new keywords for setup strings, or they may alter the meaning of some setup keywords.

Valid keywords, their proper values and their meanings, are explained in IBM's WPS documentation on the Developer Connection CD-ROM. An overview of the various object classes is also provided. The keywords and values for the WPS object classes used in our REXX examples are included in this article and are indicated with *two* leading asterisks in Figure 1.

Most keywords are found in the object class `WPObj`, (Figure 2). Much interesting information can be stored with WPS objects. For example, the

<u>KEYNAME</u>	<u>VALUES</u>	<u>REMARKS</u>
CCVIEW	DEFAULT, YES, NO	Create new view on user open ?
DEFAULTVIEW	SETTINGS, DEFAULT, id	Set defaultview on user open.
HELPLIBRARY	filename	Set the help library.
HELPPANEL	id	Set help id.
HIDEBUTTON	YES, NO	Use hide-button on window?
ICONFILE	filename	Set icon.
ICONPOS	x,y	Set icon position in folder in percent coordinates.
ICONRESOURCE	id,module	Set DLL and icon ID within.
MINWIN	HIDE, VIEWER, DESKTOP	Determine where to minimize.
NOCOPY	YES, NO	Copy of object not allowed?
NODELETE	YES, NO	Deletion of object not allowed?
NODRAG	YES, NO	Dragging of object not allowed?
NODROP	YES, NO	Dropping on object not allowed?
NOLINK	YES, NO	Creating links of object not allowed?
NOMOVE	YES, NO	Moving of object not allowed?
NOPRINT	YES, NO	Printing of object not allowed?
NORENAME	YES, NO	Renaming of object not allowed?
NOSETTINGS	YES, NO	Opening of settings page not allowed?
NOSHADOW	YES, NO	Same as NOLINK
NOTVISIBLE	YES, NO	Object not visible on desktop?

Figure 2. The setup string parameters for `WPObj`. (Continued on page 44.)

OBJECTID	<name>	Any unique string preceded with a "<" and terminated with a ">".
OPEN	SETTINGS, DEFAULT	Opening settings view or default view.
TEMPLATE	YES, NO	Object is a template?
TITLE	Title	Set title of the object.

Figure 2. The setup string parameters for WPObject. (Continued from page 43.)

REXX-FUNCTION	Win*-API-CALL
Querying, installing and removing WPS object classes:	
SysQueryClassList	WinEnumObjectClasses()
SysRegisterObjectClass	WinRegisterObjectClass()
SysDeregisterObjectClass	WinDeregisterObjectClass()
Creating, manipulating, deleting WPS objects:	
SysCopyObject	WinCopyObject()
* SysCreateObject	WinCreateObject()
SysCreateShadow	WinCreateShadow()
* SysDestroyObject	WinDestroyObject()
* SysMoveObject	WinMoveObject()
SysOpenObject	WinOpenObject()
SysSaveObject	WinSaveObject()
* SysSetObjectData	WinSetObjectData()

Figure 3. The REXX WPS interface functions are passthru of their corresponding Win-API-calls.

```

/* WPS_QCLS.CMD: query and display installed WPS object classes */

/* load OS/2's REXXUTIL functions, if not loaded already */
IF RxFuncQuery('SysLoadFuncs') THEN
DO
  CALL RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
  CALL SysLoadFuncs
END

CALL SysQueryClassList "ObjCls." /* get a list of object classes */
CALL sort /* sort in alphabetic order */

SAY "The following WPS object classes are installed:"
DO i = 1 TO ObjCls.0
  PARSE VAR ObjCls.i classname classDLL
  SAY RIGHT(i, 4) LEFT(classname " ", 35, ".") classDLL
END
EXIT

/* one of Knuth's algorithms; sort object classes in stem ObjCls. */
SORT: PROCEDURE EXPOSE ObjCls.
  M = 1 /* define M for passes */
  DO WHILE (9 * M + 4) < ObjCls.0
    M = M * 3 + 1
  END

```

Figure 4. Querying installed WPS object classes. (Continued on page 45.)

key NODELETE determines whether or not a specific object can be deleted, depending on whether it is YES or NO. The key OBJECTID provides a user-defined unique identity for each object, and that string must be enclosed between brackets: "<" and ">".

FORMATTING OF SETUP STRINGS

A setup string contains a *keyword*, immediately followed by an equal sign (=), then followed by the *value(s)*, and then the mandatory *semicolon* (;) that concludes a keyword/value pair. Usually, if a keyword accepts a list of values, those values are delimited by a simple comma (,).

In the case that you need to use the special characters comma or semicolon as part of a string value, you need to escape them with a *caret* (^), such as ";," or ";^;".

As a rule, the values are always interpreted by the objects themselves (methods defined for that object class), depending on the keywords with which they occur.

To learn even more about setup strings, one could take a look into the ASCII files named \OS2*.rc and \OS2\INSTALL*.rc on the boot drive, which the OS/2 install program uses to create the initial desktop. As a matter of fact, you should be able to write a REXX program that uses these files to recreate the standard desktop after reading this article.

INTERACTING WITH THE WPS

In order to use WPS REXX functions, you need to load them from OS/2's REXXUTIL.DLL. It is a good idea to have all of the REXXUTIL functions available at all times, as they provide many useful interfaces to OS/2 specifics.

The REXX interface functions for interacting with the OS/2 Warp WPS are listed in Figure 3. The REXX functions used in these examples are denoted by a leading asterisk.

QUERYING INSTALLED WPS OBJECT CLASSES

WPS object classes are implemented in the form of DLLs. Anyone can add new WPS object classes or even replace the existing ones. Figure 4 demonstrates the use of SysQueryClassList to query all installed WPS object classes.

First, the program checks to see whether the REXXUTIL functions are already loaded and loads them if necessary. Then, the result of the SysQueryClassList gets stored into a REXX-array with the supplied stem by the name of ObjCls. All entries get


```

DO WHILE M > 0                /* sort stem */
  K = ObjCls.0 - M
  DO J = 1 TO K
    Q = J
    DO WHILE Q > 0
      L = Q + M
      /* make comparisons case-independent */
      IF TRANSLATE(ObjCls.Q) <= TRANSLATE(ObjCls.L) THEN
        LEAVE
      tmp = ObjCls.Q /* switch elements */
      ObjCls.Q = ObjCls.L
      ObjCls.L = tmp
      Q = Q - M
    END
  END
  M = M % 3
END
RETURN

```

Figure 4. Querying installed WPS object classes. (Continued from page 44.)

```

/* WPS_NDEL.COM: make all WPS-default objects non-deletable */

/* query all WPS-OBJECTID's (OS2.INI, app; "PM_Workplace:Location") */
CALL SysIni "USER", "PM_Workplace:Location", "ALL:", "object_id"

leadin_string = "<WP_" /* work on WPS-objects only */
setup = "NODELETE=YES;" /* setup string: change to not deletable */
SAY "Trying to make all WPS-objects non-deletable:"; SAY

DO i = 1 TO object_id.0 /* loop over all entries */
  IF ABBREV(object_id.i, leadin_string) THEN /* WPS-OBJECTID ? */
    DO
      ok = SysSetObjectData(object_id.i, setup) /* change state */
      SAY LEFT(object_id.i || " ", 57, ".") WORKED(ok)
    END
  END
END
EXIT /* end of program */

/* procedure to indicate successful/not successful */
WORKED: PROCEDURE
  IF ARG(1) THEN RETURN "successful."
  ELSE RETURN "*** NOT succesful ***"

```

Figure 5. Making default WPS objects nondeletable.

sorted into ascending order and then the WPS object class name gets parsed from the DLL filename from which that object class is implemented. The output is formatted for easier reading.

MAKING ALL WPS-DEFAULT OBJECTS NONDELETABLE

When the WPS gets installed, there are many default folders and objects created on the desktop, such as the system folder, the enhanced EPM editor, and so forth. Look at the file \OS2\INI.RC, which drives this setup.

All of these standard WPS-objects get a unique OBJECTID, which starts with the string <WP_. An OBJECTID is a

string starting with a < and ending with a >. Knowing the OBJECTID of an object lets you directly set its properties, delete it, move it, and produce links (shadows) of it. All user-defined OBJECTIDs of the WPS are stored in the OS2.INI file with the application name PM_Workplace:Location.

Be aware though, that it is possible to create objects without an explicit, user-defined OBJECTID. In such cases, one cannot get a handle to the object from within REXX.

Figure 5 first gets all known user-defined OBJECTIDs by using the REXXutil function SysIni to access the OS2.INI file, including all entries stored with

OS/2® and AIX Tools at Your Command

The best way to find out what's available for IBM system software

platforms.

Stay on top of it by browsing this software and services directory of products for OS/2, AIX and other IBM supported operating systems.

Sponsored by IBM®, OS/2 Developer, OS/2 Magazine, and Software Development.

OS/2® is a registered trademark of International Business Machines Corporation and is used by Miller Freeman, Inc. under license.

Find It on the Web!


```

/* WPS_TITL.CMD: show setting a title */

/* setting a title with SysCreateObject */
title = "This is a title,"this is the second line;"this the third !"
objectid = "<RGF Testfolder for title>"
setup_string = "ICONPOS=15,30;OBJECTID=" || objectid || ";";

ok = SysCreateObject("WPFolder",, /* instance of WPFolder */
                    title,, /* object title */
                    "<WP_DESKTOP>",, /* location: desktop */
                    setup_string,, /* setup string */
                    "F") /* fail, if object exists */
SAY "creating:" objectid "-" worked(ok)
"@PAUSE"

/* note how to escape a semi-colon which usually ends a key-value */
setup_title = "This is a title,"this is the 2nd line";"this the 3rd!"

setup_string = "TITLE=" || setup_title || ";";
ok = SysSetObjectData(objectid, setup_string) /* change title */
SAY "changing title:" objectid "-" worked(ok)
"@PAUSE"

SAY "cleaning up..."
ok = SysDestroyObject(objectid) /* delete folder */
SAY "destroying (deleting):" objectid "-" worked(ok)

EXIT

/* procedure to indicate successful/not successful */
WORKED: PROCEDURE
    IF ARG(1) THEN RETURN "successful."
    ELSE RETURN "*** NOT succesful ***"

```

Figure 6. Setting an object title at creation time and with SysSetObjectData.

the application key PM_Workplace: Location. Then every OBJECTID is checked to see if it starts with the string <WP_. If it does, it uses the REXXUtil function SysSetObjectData to change the state of that object to nondeletable. This is done by passing the setup string NODELETE=YES; to the object identified by the OBJECTID. This property is defined for every single object of type WPObject (Figure 2).

From this moment, the default WPS-objects cannot be deleted, and the delete option on the object pop-up menu is missing. To revert the setting, change the setup string to NODELETE=NO; and run the program again.

SETTING THE TITLE OF AN OBJECT

A title for a WPS object may be up to 255 characters long at present. In the case where a title contains line breaks, this will be indicated with a caret (^) right before the intended line break in the setup string.

There are two places one can define a title for a WPS object with REXX: at object creation time (SysCreateObject) and using SysSetObjectData.

When using the TITLE keyword in the setup string to name the title, do not forget that a semicolon ends a key/value pair, so you need to escape



Invest a stamp Save a bundle

For the price of a stamp, you can get the latest edition of the federal government's free **Consumer Information Catalog** listing more than 200 free or low-cost govern-

ment publications on topics such as federal benefits, jobs, health, housing, education, cars, and much more. Our booklets will help you save money, make money, and spend it a little more wisely.

So stamp out ignorance, and write today for the latest free **Catalog**. Send your name and address to:

**Consumer Information Center
Department SB
Pueblo, Colorado 81009
U.S. General Services Administration**

A public service of this publication and the Consumer Information Center of the U.S. General Services Administration.

KEYNAME	VALUES	REMARKS
ALWAYSSORT	YES, NO	Sort always?
BACKGROUND	N,M,S,B,C	Set folder background, characters in this list represent:
	N	Image file name.
	M	Image mode, one of three characters: N (normal), T (tiled) or S (scaled).
	S	Scaling factor.
	B	Background type, one of two characters: I (image) or C (color only)
	C	Background color in RGB notation.
		Example with a scaled bitmap: "BACKGROUND=D:\BITMAP\WARP.BMP,S,3,I"
		Example with cyan-like color background: BACKGROUND=(none),,,C,28 233 233"
DEFAULTVIEW	ICON, TREE, DETAILS	Default view of container .
DETAILSCLASS	classname	Set object class for details.
DETAILSFONT	size.font_name	Set font for detail view (10 Helv. for example).
DETAILSVIEW	s1[,s2,...sn]	Set details view to ... (see WPS docs)
ICONFONT	size.font_name	Set font for icon view (10 Helv. for example).
ICONNFILE	index,filename	Set "open" icon, index is always 1.
ICONNRESOURCE	index,id,modname	Set "open" icon, icon ID within DLL; index is always set to 1.
ICONVIEW	s1[,s2,...sn]	Set icon view styles (see below).
ICONVIEWPOS	x,y,cx,cy	Initial view position and size of open folder in percentage coordinates.
OPEN	ICON, TREE, DETAILS, DEFAULT	Open folder in respective view.
REMOVEFONTS	YES, NO	Remove instance fonts from folder?
SORTCLASS	classname	Set class object to sort by.
TREEFONT	size.font_name	Set font for tree view (10 Helv. for example).
TREEVIEW	s1[,s2,...sn]	Set tree view styles (see below).
WORKAREA	YES, NO	Set folder to be a workarea?

Figure 7. The Setup String parameters for WPFolder. (Continued on page 48.)

the semicolon with a caret (^) if it is to appear in the title's text. Note that in the TITLE keyword, the comma in the string has no special meaning.

Figure 6 demonstrates both scenarios. First we have to create a WPS folder object from the WPFolder object class having the properties of any WPS folder. This object gets the OBJECTID <RGF Testfolder for title>, so we can address it later from REXX. The icon for the folder gets placed 15% right from the left screen margin and 30% up from the bottom of the screen (ICONPOS=15,30;). Because of the per-

centage coordinates, we do not have to get involved with physical screen resolutions.

If an object with the given OBJECTID exists already, SysCreateObject will fail. Other options would be UPDATE or REPLACE. If you leave the OBJECTID out of the setup string, the WPS will create a new object.

If the user presses the return key, the title of this folder gets updated via SysSetObjectData. After another press of the return key, it is deleted. Compare the use of the caret (^) in the two title strings.

CREATING FOLDERS, SHADOWS AND MOVING OBJECTS

WPS folders gain additional setup strings, which are depicted in Figure 7, of which we will use the key ICONVIEWPOS in Figure 8. ICONVIEWPOS allows us to define a default starting location *relative* to the lower left corner of the screen and, in addition, the dimension (width, height) of the open folder. All numbers are percent coordinates.

There are two ways to create a shadow of an object, but both need the knowledge of the OBJECTID of the object to be shadowed:

An Internet-based Directory to Thousands of Products and Services

OS/2® & AIX
developers,
systems
administrators,
and users: the
tools you need
are just a few
keystrokes
away.

Thousands
of product &
service listings for
IBM supported
operating systems,
continually
updated, easy
to search.

Sponsored by IBM®, OS/2 Developer,
OS/2 Magazine, and Software Development.

OS/2® is a registered trademark of
International Business Machines
Corporation and is used by Miller
Freeman, Inc. under license.

**Find It
on the
Web!**

<http://www.mfi.com/softwareguide>

VIEW STYLES	DESCRIPTION
FLOWED	Flowed icon view.
NONFLOWED	Non-flowed icon view.
NONGRID	Non-gridded icon view.
NORMAL	Normal size icons.
MINI	Small icons.
INVISIBLE	No icons.
LINES	Lines in tree view.
NOLINES	No lines in tree view.

You may combine view styles by delimiting the view style values with a comma. For example, "ICONVIEW=FLOWED,MINI;"

Figure 7. The Setup String parameters for WPFolder. (Continued from page 47.)

```

/* WPS_TSTF.CMD: test folders, shadows and moving objects */

/***** create four folders on the desktop to play with *****/
DO i = 1 TO 4
  objectid = "<RGF Testfolder_" || i || ">" /* unique objectid */
  title = "Testfolder #" i
  folder.i = objectid /* save for later use */
  y_pos = 100 - i*20 /* placement on y-axis */
  setup = "ICONPOS=5," || y_pos || ";" || /* icon placement */
  "ICONVIEWPOS=15," || y_pos || /* initial folder dim: */
  ",35,20;" || /* relative to icon */
  "OBJECTID=" || objectid || ";" /* assign objectid */

  ok = SysCreateObject( /* create object */
    "WPFolder", /* Object type: WPS-Folder */
    title, /* Title */
    "<WP_DESKTOP>", /* create on desktop */
    setup, /* setup-string */
    "F") /* fail, if object exists */

  SAY "creating:" objectid "-" worked(ok)
END

folder.0 = 4 /* indicate 4 elements in array */
SAY /* display empty line */

/***** change state-data to open folder in icon view *****/
"@PAUSE"
DO i = 1 TO 3 /* open first three folders */
  setup = "OPEN=ICON;" /* open them using icon view */
  ok = SysSetObjectData(folder.i, setup)
  SAY "setup object data ["setup"] for" folder.i "-" worked(ok)
END

/***** create shadow of OS/2 Configuration Folder *****/
shadObjID = "<RGF Testshadow_1>" /* OBJECTID of shadow */
folder.5 = shadObjID /* memorize */
folder.0 = 5 /* now we have 5 elements */
setup = "SHADOWID=<WP_CONFIG>;" || /* shadow OS2-config folder */
"OBJECTID=" || shadObjID || ";" /* OBJECTID of shadow */

```

Figure 8. Creating folders, a shadow and moving objects. (Continued on page 49.)


```

ok = SysCreateObject(,          /* create object          */
  "WPSshadow",,              /* Object type: WPS-Shadow */
  title,,                    /* Title                  */
  folder.1,,                 /* put into folder # 1    */
  setup,,                     /* setup-string           */
  "F")                        /* fail, if object exists */
SAY "creating shadow for" folder.5 "-" worked(ok)
SAY                             /* display empty line    */

/***** move Testfolder # 4 between the first three folders *****/
location1 = 1                 /* shadow in folder #1    */
location2 = 0                 /* folder #4 on desktop   */

DO FOREVER
  SAY "Press enter to animate (enter 'exit' to end):"
  PARSE UPPER PULL input      /* get input from user    */
  IF LEFT(input, 1) = "E" THEN LEAVE /* just check first letter */

  location1 = (location1 // 3) + 1 /* next folder for shadow */
  location2 = (location2 // 3) + 1 /* next folder for folder # 4 */

  ok = SysMoveObject(shadObjID, folder.location1) /* move shadow */
  SAY "moving shadow" shadObjID "to" folder.location1 "-" worked(ok)
  ok = SysMoveObject(folder.4, folder.location2) /* move folder */
  SAY "moving folder" folder.4 "to" folder.location2 "-" worked(ok)
  SAY                             /* display empty line    */
END

/***** delete test folders ? (delete, if first letter is "Y" *****/
SAY "Delete test folders ? (Yes/No)"
PARSE UPPER PULL input
IF LEFT(input, 1) = "Y" THEN /* delete test folders ? */
DO i = folder.0 TO 1 BY -1 /* delete, start with # 5 */
  ok = SysDestroyObject(folder.i)
  SAY "destroying (deleting):" folder.i "-" worked(ok)
END

EXIT

/* procedure to indicate successful/not successful */
WORKED: PROCEDURE
  IF ARG(1) THEN RETURN "successful."
  ELSE RETURN "*** NOT successful ***"

```

Figure 8. Creating folders, a shadow and moving objects. (Continued from page 48.)

- SysCreateObject ("WPSshadow", ...)
- or
- SysCreateShadow (objectid, target folder).

Using the WPS object class, WPSshadow lets us define an explicit OBJECTID for the shadow object itself. This WPS object class adds the keyword SHADOWID to the setup string, which allows the OBJECTID of the object to be shadowed assigned.

A simpler way to achieve the same effect is to use SysCreateShadow, but in this case you cannot assign the shadow object an OBJECTID of your choice.

Figure 8 creates four folders and determines the initial positions for the test folder icons (ICONPOS) and the posi-

tion and size for opening those folders (ICONVIEWPOS). Using SysSetObjectData, the first three folders get opened in icon view. Then, a shadow of the OS/2 configuration folder (<WP_CONFIG>) is placed into the first of the open folders. By using SysCreateObject, it is possible to assign the shadow object an OBJECTID of its own, which we will need in order to move it from one open test folder to the next.

The moving (animation) of the shadow and the fourth test folder occurs as long as the user presses the enter key: the shadow object gets moved to the next open test folder, as well as the object <RGF Testfolder_4> (obviously we use the function SysObjectMove to move the objects). It

may look as if the fourth test folder object chases the shadow object through the first three open test folders.

Last, the shadow and test folders may get deleted (SysDestroyObject) from the WPS at the user's request.

CREATING A CUSTOMIZED DOS PROGRAM FROM NOWHERE

Our last example demonstrates how to define and create objects of type WPPProgram. This object class adds quite a few additional setup strings (Figure 9). With WPPProgram object classes, we are able to produce any type of program capable of running on OS/2. Especially for DOS and WIN-OS2 programs, it is very convenient to set the characteristics of the virtual machines (virtual PCs) those programs run in.

Figure 10 demonstrates the set up of a windowed virtual DOS machine (PROGTYPE=WINDOWEDVDM;), running the DOS program \OS2\MDOS\MEM.EXE. Note that the keywords EXENAME and STARTUPDIR of WPPProgram allow for substituting the boot drive for the string ?:\. In Figure 10 the EXENAME is merely a star (*), indicating a plain DOS-, Windows- or OS/2-session. In this case, the setup key, PROGTYPE, determines the session type to be DOS; the program MEM.EXE gets started because of the setup string PARAMETERS=/k mem.exe;. The DOS-session window remains open because of /k, which tells COMMAND.COM to keep the session after the program (mem.exe) ran.

Most of this REXX program deals with configuring the virtual machine, so the setup string consists of multiple SET some_keyword=...; statements. You can name any setting keyword from the settings page of DOS program objects. If a DOS setting like DOS_UMB has just two options, like ON or OFF, you would use 1 resp. 0. Otherwise, you use string values of the various DOS settings verbatim. Multiple entries for DOS_DEVICE and DOS_VERSION are separated with a comma.

Do not forget to escape commas and semicolons with the caret (^) if they should become part of a setup string for a DOS setting (see the DOS setting for DOS_VERSION in Figure 10).

One interesting feature of Figure 10 is hidden in the location for the WPPProgram object: <WP_NOWHERE>. This location does not exist in a visible form on the desktop (it is hidden), hence, the name nowhere. One could use this location for dynamically built objects that should not be seen by the user. In Figure 10, we use the same OBJECTID to open (start) the WPPProgram object multi-

KEYNAME	VALUES	REMARKS
ASSOCFILTER	filters	Set filename filters.
ASSOCTYPE	type	Set association types.
EXENAME	filename	Program to be executed.
MAXIMIZED	YES, NO	Maximized upon startup?
MINIMIZED	YES, NO	Minimized upon startup?
PROGTYPE	FULLSCREEN PM PROG_31_ENH PROG_31_ENHSEAMLESSCOMMON PROG_31_ENHSEAMLESSVDM PROG_31_STD PROG_31_STDSEAMLESSCOMMON PROG_31_STDSEAMLESSVDM SEPARATEWIN VDM WIN WINDOWABLEVIO WINDOWEDVDM WINDOWEDWIN	Full-screen OS/2 session. PM application. WINOS2 3.1 enhanced mode. WINOS2 3.1 enhanced mode, seamless common session. WINOS2 3.1 enhanced mode seamless session. WINOS2 3.1 standard mode. WINOS2 3.1 standard mode, seamless common session. WINOS2 3.1 standard mode, seamless session. WINOS2 session in separate VDM. Full screen DOS session. Full screen WINOS2 session. Windowed OS/2 session. Windowed DOS session. Windowed WINOS2 session.
NOAUTOCLOSE	YES, NO	Do not close window upon program! termination?
PARAMETERS	params	Set program parameters.
SET	XXX=VVV	Set environment variables. Set VDM characteristics for DOS sessions.
STARTUPDIR	pathname	Set the working directory.

Figure 9. The Setup String parameters for WPProgram.

```

/* WPS_NOWH.CMD: demo setting and starting DOS-programs from nowhere*/

/* change the location to "<WP_DESKTOP>" to check the settings-page */
location = "<WP_NOWHERE>" /* the "NOWHERE"-location */

title = "Test of a DOS-Program from nowhere :-)"
objectid = "<RGF TEST_DOS_IN_NOWHERE>"
setup = "PROGTYPE=WINDOWEDVDM;" ||,
        "EXENAME=;" ||,
        "PARAMETERS=/k mem.exe;" ||,
        "STARTUPDIR=?\;" ||,
        "SET COM_RECEIVE_BUFFER_FLUSH=SWITCH TO FOREGROUND;" ||,
        "SET DOS_DEVICE=\os2\mdos\ANSI.SYS;" ||,
        "SET DOS_VERSION=NETX.EXE,5,00,255," ||,
        "WIN200.BIN,10,10,4;" ||,
        "SET DOS_UMB=1;" ||,
        "SET EMS_MEMORY_LIMIT=0;" ||,
        "SET IDLE_SECONDS=1;" ||,
        "SET IDLE_SENSITIVITY=10;" ||,
        "SET XMS_MEMORY_LIMIT=4096;" ||,
        "OPEN=DEFAULT;" ||,
        "OBJECTID=" || objectid || ";"

```

Figure 10. Creating and starting a customized DOS-program from nowhere. (Continued on page 51.)

ple times, so we use the REPLACE option on the call to SysCreateObject. In case you are interested whether the DOS settings have the values expressed in Figure 10, merely replace the location in SysCreateObject with <WP_DESKTOP> and inspect the settings page of it, once the WPProgram object has been created.

A LAST WORD ON OBJECTIDS

Whenever it is possible to supply a full path for an object (files that are objects or directories which are objects representing folders), these full paths can be given instead of a WPS OBJECTID.

Hence, if your boot drive was C:, and you have installed the file "Enhanced Editor for PM" (EPM.EXE), you could use the predefined WPS OBJECTID <WP_EPM> or the C:\OS2\APPS\EPM.EXE instead, the physical location of EPM.EXE on your drive C:.

Under the application key PM_WorkPlace:Location in the OS2.INI file,


```

ok = SysCreateObject("WPPProgram", title, location, setup,,
                    "R")      /* replace, if object exists */

SAY "creating:" objectid "-" worked(ok)
"@PAUSE"

SAY "Cleaning up ..."
ok = SysDestroyObject(objectid) /* OBJECTID of object */
SAY "destroying (deleting):" objectid "-" worked(ok)

EXIT

/* procedure to indicate successful/not successful */
WORKED: PROCEDURE
  IF ARG(1) THEN RETURN "successful."
  ELSE RETURN "*** NOT succesful ***"

```

Figure 10. Creating and starting a customized DOS-program from nowhere. (Continued from page 50.)

only user-defined OBJECTIDs of the form <some_unique_string> are stored. All WPS objects get object handles assigned by the WPS. PM_Workplace:Location simply serves as a directory that maps user-defined OBJECTIDs to their WPS object handles. Therefore, objects created without an OBJECTID given in the setup string for SysCreateObject would not appear in this list. The same is true for

SysCopyObject and SysCreateShadow.

Rony G. Flatscher is an assistant professor in the Department of Management Information Systems, Vienna University of Economics and Business Administration, Vienna, Austria. He heads the Austrian SIG OS/2 Forum Austria, which he helped found. He can be reached at e-mail Rony.Flatscher@wu-wien.ac.at or by telephone (+43) - 1 - 313 36 extension 4881.

REFERENCES

Many thanks to Rick McGuire, Georg Haschek, and Mike Lamb from IBM for sharing their WPS-related REXX programs, hints, and tricks with the public.

Check out the ftp sites <ftp://hobbes.nmsu.edu> or [os2.cdrom.com](ftp://os2.cdrom.com) for REXX programs/docs dealing with the WPS, such as [cr-tobj.zip](#) or [rexxtut.zip](#). A good place on the internet is the newsgroup [comp.lang.rexx](#), which gets occasional visits from M.F. Cowlishaw (inventor of REXX) and Rick McGuire (lead designer for the soon to be released follow-up to REXX—Object REXX).

To search WPS- and REXX-related programs and docs on the internet ftp-sites, you may also use the URL <http://www.os2forum.or.at/software/htmlmirrors/> entering WPS or REXX and choosing the appropriate ftp servers.

☐ **YES! I want to receive/continue to receive Network VAR, FREE!**

☐ No

Signature ☒ _____ Date _____
 Name _____ Title _____
 Company _____
 Street _____ P.O. Box _____
 City/State/Zip _____
 Business Phone () _____
 FAX () _____
 If company policy requires home delivery, please complete home address. Company name and title are still needed to process form.
 Home Street _____
 Home City _____
 State/Country _____
 Zip/Postal Code _____

The publisher reserves the right to determine eligibility for free subscriptions.

1 Are you involved in the value added reselling and/or integration of networking systems, software or services? (check only one)

☐ 001 Yes ☐ 002 No

2 Which best describes your firm's primary business? (check only one)

☐ 101 Network VAR/Reseller
☐ 102 Systems Integrator/Network Integrator
☐ 103 Independent Consultant
☐ 104 In-house Corporate Integrator

☐ 105 Manufacturer who Resells
☐ 106 Distributor
☐ 149 Other _____ (please specify)

3 Which best describes your job title? (check only one)

☐ 151 Executive Management (Chairman CEO/President/Owner/Principal Partner)
☐ 152 General Management (VP Finance/Operations/Department Head/General Manager/Purchasing)

☐ 153 Technical Management (VP Technical Services/Chief Technical Officer/Engineering Director)

☐ 154 Technical Staff (Systems Analyst/Systems Engineer)

☐ 155 Sales Management (VP Sales/Sales Director)

☐ 156 Sales Staff (Sales Engineer/Account Manager)

☐ 157 Consultant

☐ 199 Other _____ (please specify)

NetworkVAR

NETWORK SOLUTIONS FOR THE CHANNEL

The time has come!

Take a minute and unlock a year's worth of free comprehensive technical material on the entire spectrum of network technology by completing this subscription form.

Network VAR is the only magazine dedicated 100% to the special information needs of network VARs and integrators.

P. O. Box 469050 • Escondido, CA 92046-9050 • or FAX: 415-905-4967

H5601



Buyers Guide

The staff of OS/2 Developer put together this special section to guide you through the various REXX products available for OS/2. The products in this guide are based on surveys sent to vendors; they are provided as a service to you and are free to vendors. These listings do not represent an endorsement by OS/2 Developer. Although every effort has been made to ensure accuracy, we do not assume any responsibility for error or omission in this section. Please contact the companies directly for more information.

REXX Buyers Guide

DB TECHNOLOGIES INC.

Entry Field Object Pack for VX-REXX 2.1.3 formats entry fields for date, time, telephone, social security number, and zip code. A picture object is included for special programmer-defined formats. Compatible with VX-REXX 2.1 or higher, this product can be bound to database columns via the VX-REXX Client/Server Query Object. Price: \$49.95.

Multimedia Object Pack for VX-REXX 1.0.1 provides objects for the production of professional OS/2 multimedia applications. It features Animated Graphic Pushbuttons, Circular Slider, and MCI Object. In addition, programmers can control any device manipulated via the MCI String command interface. This package provides real-time synchronous event notification and is compatible with VX-REXX 2.0 or higher. Price: \$99.

VX-Calendar Object 1.0.1 is a programmable calendar object for Watcom VX-REXX developers. Features include drag-and-drop support of date information, programmable date ranges, changeable day-of-week ordering on the calendar's face, and multiple font and color support. It is compatible with VX-REXX 2.0 or higher. Price: \$99.

Enhanced Picture Box Object for VX-REXX 1.0.0 provides the functionality of the standard VX-REXX Picture Box and adds two properties, MouseXPos and MouseYPos, that continuously report current mouse pointer position. Every mouse move on the object generates a MouseMove event. This product is compatible with VX-REXX 2.0 or higher. Price: \$19.95.

Progress Indicator Object for VX-REXX 1.0.0 is an object for projects requiring progress indicator bars for file transfers and other lengthy procedures. This product features horizontal

and vertical sizing, multiple font and color support, and optional percent text display. It is compatible with VX-REXX 2.0 or higher. Price: \$29.95.

DB Technologies Inc. 2420 Briar Oak Circle, Sarasota, Fla. 34232, (800) 830-8703 or (813) 379-8766, fax (813) 377-7392.

HOCKWARE INC.

VisPro/REXX Gold edition 2.11 is a drag-and-drop visual programming tool for 32-bit OS/2 2.x GUI development. VisPro includes all of CUA '91 objects plus 3D business graphics and multimedia, full Workplace Shell integration, multiple threads support, visual DB2/2 database designer, multiple development views, SOM-based object builder, and it provides standalone, royalty-free executable. LAN version is also available. Price: \$299.00.

VisPro/REXX Bronze edition 2.11 provides programmers with an easy-to-use drag-and-drop environment to facilitate development for 32-bit OS/2 2.x GUI applications. It includes many of the CUA '91 objects, full Workplace Shell integration, multiple development views, and SOM-based object builder, and it provides standalone, royalty-free executable. An upgrade to the Gold edition is available. Price: \$59.

Vis/Pro REXX Data Entry Object Pack 1.01 is an add-on package of five popular business objects for both Vis/Pro REXX Gold and Bronze editions. Objects include formatted entry field, spreadsheet, split bar, calendar, and clock. Price: \$89.

VisPro/Reports 1.01 is a REXX-enabled report writer for VisPro/REXX, VX REXX, and OS/2 REXX. It allows you to print invoices, form letters, personnel reports, financial

reports, and so on. A wide variety of Avery labels are supported, allowing you to print addresses and other labels. A CUA '91 style WYSIWYG report designer allows you to include business graphics, photographs, calculated fields, multiline text, and other cosmetic shapes. Apply numerous object attributes, including 16 million colors, OS/2 ATM fonts, line styles, line widths, fill patterns, and drop shadows. Price: \$199.

VisPro Development Suite 2.11 includes VisPro/REXX Gold, VisPro/C, and VisPro/C++. Price: \$499.

Hockware Inc., 315 North Academy St., Ste. 100, Cary, N.C. 27513, (919) 380-0616, fax (919) 380-0757.

LESTEC PTY LTD.

Modular and Integrated Design 1.2 provides a WYSIWYG graphic editor for prototyping and designing your own ready-to-run PM interface producing simple reusable dialog scripts (modules). Modules integrate into the OS/2 Desktop and support REXX scripts, multithreading, interdialog communication, and control for your environment. External C, REXX interface, and support for database and external communications provided. Price: \$199.

LesTec PTY Ltd., P.O. Box 1394, Dee Why, NSW, 2099, Australia 61-2-998-15925, fax 61-2-998-15925.

MANSFIELD SOFTWARE GROUP INC.

KEDIT 5.0 for OS/2 is a general-purpose text editor that uses REXX as its macro language. KEDIT, a 32-bit text mode application, is compatible with IBM's XEDIT editor. Features include selective line editing, column editing, undo/redo, file locking, online help, SORT command, long filename support, and multiple files and windows. Price: \$189.

Mansfield Software Group Inc., P.O. Box 532, Storrs, Conn. 06268, (203) 429-8402, fax (203) 487-1185.

NOMBAS

CEnv 2.01 for OS/2 implements the Cmm scripting language to enhance batch REXX files, create simple personal utilities, and automate program execution. Features include full interaction with OS/2 commands, API DLLs, and environment variables; concise documentation; extensive .CMD code library; 100 examples; compact executable size; and keyboard, menu, dialog, and clipboard automation. Price: \$45.

Nombas, 64 Salem St., Medford, Mass. 02155, (617) 391-6595, fax (617) 391-3842.

QUERCUS SYSTEMS

Personal REXX 3.0 augments OS/2 REXX by including REXXLIB extensions for array handling, mathematical functions, interprocess communication, and gen-

eral system services. In addition, it provides global variables, additional utilities for VM/CMS and MVS/TSO compatibility, and enhanced external data queue support. Price: \$175.

REXXLIB 1.0 includes over 150 REXX extension functions for compound variable handling, interprocess communication, mathematics, system services, and text mode user interfacing. Functions include date format conversion, retrieval of compound variable "tails," sorting and other array operations, "regular expression" searches, and reading or writing collections of variables in the disk files. There are no run-time fees. Price: \$50.

REXXTERM 2.3 is a full-featured asynchronous communications program that provides extensive support for user customization and script writing with REXX. Other major features include multiple dialing directories, built-in text editor, host mode, flexible keyboard configuration, VT102 and VT52 terminal emulation, and Xmodem, Ymodem, Zmodem, Kermit, and CompuServe-B file transfer protocol. Price: \$100.

Quercus Systems, P.O. Box 2157, Saratoga, Calif. 95070, (800) 440-5944 or (408) 867-7399, fax (408) 867-7489.

STRATEGIC SOLUTIONS INTERNATIONAL CORP.

Service Point/32 1R8 is a fully featured, bidirectional service point providing alert generation and RUNCMD processing facilities from REXX and C. REXX scripts developed on OS/2 platforms are interchangeable with host NetView environment for RUNCMD processing. Terminal emulation for VT-100, ANSI 3.64, and IBM 3164 enable REXX automation to communicate with outbound management systems and generate network management status via alerts to NetView, NetMaster, and/or NetImpact. 3270 Panel interface provided in OS/2 REXX emulates NetView's REXX Panel interface for development of user-friendly, full-screen interactive application development to front end automated commands. Corporate licensing is available. Price: \$3,000.

NetImpact 3R1, an OS/2-based PM/Workstation, provides a GUI for MVS-based SNA/Agent that auto-discovers network and reports availability and reliability of managed elements. OS/2 REXX interface provides automation facilities for VTAM and MVS commands to address problems with SNA network and with MVS applications. Pricing is based on SNA/Agent which includes licenses for PM/Workstations with REXX automation facilities. It does not require NetView, but will run if NetView is installed. Price: \$20,000 (dependent on MVS/ESA CPU MIPS).

Strategic Solutions International Corp., 1127 Tolland Tnpk., Manchester, Conn. 06040, (203) 649-1900, fax (203) 649-1230.



VTs-DATENSYSYSTEME GMBH & CO.

REXXLAN/2 2.0 combines the information quantity of the LAN Server C-API with the advantages of REXX programming. Application fields include creation of reports from domain controller databases according to specified criteria, automation of domain migrations, management of print queues, and automated user setup. If server applications are to be created with the native C-API, REXXLAN/2 allows prototyping and subsequent simple code conversion to the final application with its identical representation of the data structure and functions. Price: about \$400.

VTs-Datensysteme GmbH & Co., P.O. Box 305583, D-20317 Hamburg, Germany, 49-40-656-2856, fax 49-40-657-1218.

LOTUS DEVELOPMENT CORP.

Ami Pro for OS/2 takes advantage of OS/2's multithreading power and delivers seamless integration with the Workplace Shell, providing drag-and-drop document opening, printing and shredding, as well as 55 professionally-designed style sheets for new document creation. Inline REXX command support included. Fully compatible with Ami Pro for Windows Release 3.0.

Lotus Development Corp., 55 Cambridge Pkwy., Cambridge, MA 02142, (617) 577-8500.

NOMBAS

CEnv for OS/2 is a powerful scripting tool for OS/2, Windows, and DOS. CEnv executes the CMM (C' minus minus) scripting language to enhance batch/REXX files, create new utilities, or modify and personalize existing CEnv programs written by Nombas or other CEnv users. Hundreds of sample programs demonstrate CMM programming are included. CEnv provides over 150 internal functions in addition to directly linking with all DLLs and OS/2 kernel functions.

Nombas, P.O. Box 875, Medford, MA 02115-0007, 617 391 6595.

CHRON CO.

Chron is a multithreaded PM application that allows flexible operational scheduling of workstation or server tasks and messages. Events can be scheduled One Time or every 'x' minutes, hours, days, weeks, months, or years. REXX support is added for more complex scheduling operations. Chron is now CID-enabled for easier installation.

Chron Co. Hilbert Computing, 1022 N. Cooper, Olathe, KS 66061, (913) 780-5051, fax 913-829-2450.

COMMANDLINE CO.

CommandLine is a 32-bit on-demand command prompt, allowing users to execute commands or programs without leaving the active application in search of an icon or open session, thus eliminating the irritating keyboard-mouse-keyboard cycle. The package can be invoked via a user-defined hot key, and it remembers previous commands across boot sessions and can launch DOS and windows applications. It has aliasing, inline file find, filename completion, and an inline REXX expression interpreter extendible through DLLs. CL also acts as a fast task switcher. Any program, command or object can be assigned to a hotkey, with a variety of options. A hotkey can be set to switch to an existing session or create a new one. CL can toggle between the two most recently used windows. Perfect for DEMOs! CL minimizes desktop clutter and simplifies access to WPS objects. Users can open an object by name, without opening nested folders.

CommandLine Co. Soft & GUI, Inc., 2224 East 21st St., Brooklyn, NY 11229.

COMMPAK/2 DYNAMIC LINK LIBRARY, PROGRAMMERS TOOLKIT CO.

Commpak/2 Dynamic Link Library, Programmer's Toolkit provides low-, medium-, and high-level access to the personal computer's asynchronous communications port under OS/2 for C, C++, and other high-level languages. It includes everything required for manipulating the async port via the OS/2 COM.SYS driver, plus X, Y, and ZModem file transfer protocols and a REXX interface continuously.

Commpak/2 Dynamic Link Library, Programmers Toolkit Co. Oberon Software, 1405 East Main St., Mankato, MN 56001, 507-388-7001, fax 507-388-7568.

FLEXLOAD CO.

Flexload provides an option to load data into as many DBM tables as desired simply by designating which tables are to be loaded. You may also specify the "commit" amount which will also improve the load time. You will no longer need to maintain multiple REXX programs or other load modules since Flexload will automatically rebuild your load program each

time the destination table structure changes.

Flexload Co. Peregrine Group Inc., The, 200 N. Northwest Highway, Barrington, IL 60010.

FROBOZZ INVESTOR CO.

Frobozz Investor is a personal investment application containing four modules: fund management with graphs, portfolio management with ROI calculation and graph support, technical analysis using popular indicators, and a REXX interface for data acquisition. It also has a REXX interface for external access to the database, allowing user enhancements.

Frobozz Investor Co. Frobozz Magic Software Co., The, Lange Kerkda 113, Wassenaar, 2242 BT, +31 1751 18301, fax +31 1751 11789.

GPFREXX CO.

GpFREXX combines REXX with a powerful but easy-to-use, WYSIWYG visual programming environment to make OS/2 PM programming available to almost everyone. Simply point and click to design simple utilities or complete applications. It supports standard OS/2, as well as multimedia, SQL, and advanced communications.

GpFREXX Co. Apical Software, Inc., 40 Falls Rd., PO Box 432, Moodus, CT 06469-0432, 203 873 3300, fax 203 873 3302.

IBM CORP.

IMS Client/Server/2 lets you incorporate your IMS DC or TM application base into new, PC-based business solutions. To bring your host data to the PC environment, IMS CS/2 provides an easy-to-use, callable application programming interface (API). IMS CS/2 is compatible with VisualAge, DrDialog/DrREXX, Visual C++, PL/I for OS/2 Toolkit, SmallTalk/VPM, Easel, C/C++, Micro Focus COBOL/2, OS/2 REXX, and Watcom Fortran 77.

IBM Corp., 555 Bailey Ave., San Jose, CA 95142, 800 887 7771, fax 800 342 6672.

MANSFIELD SOFTWARE GROUP INC.

KEDIT for OS/2 v. 5.0 is a powerful, general purpose text editor uses REXX as its macro language. KEDIT for OS/2, a 32-bit text mode application, is highly compatible with IBM's XEDIT editor. Features include: selective line editing, column editing, undo/redo, file-locking, on-line help, SORT command, long filename support, multiple files and windows, and more.

Mansfield Software Group Inc.,
P.O. Box 532, Storrs, CT 06268, (203)
429-8402, fax (203) 487-1185.

LESTEC PTY LTD

Modular And Integrated Design (MAID), an authoring system, represents dialogs as simple script files, providing easy access to a powerful GUI front end. It makes full use of OS/2 REXX by designing object-based dialogs as modules with object variables as controls, using event-driven REXX scripts. MAID easily integrates modules with each other and your existing programs.

LESTEC PTY LTD, P.O. Box 1394,
Sydney, 2009, 61-2-99815925, fax 61-2-
99815925.

OBERON SOFTWARE

Oberon Terminal Emulator/2 is a full-featured, easy-to-use, general purpose telecommunications program for OS/2. Features include a complete script language, optional REXX/2, 200-entry dialing directories, call logging, split-screen chat mode, and completely programmable keyboard. Supports XModem and XModem-1K, YModem and YModem-G, ZModem, CompuServe B-Plus protocol and ASCII transfer protocols.

Oberon Software, 1405 East Main
St., Mankato, MN 56001, 507-388-7001,
fax 507-388-7568.

CLEAR & SIMPLE INC.

Performance Plus V3 is a tuning and utilities kit for OS/2 WARP v.3 and 2.1. It combines 19 utilities and a 120- page book dedicated to OS/2 performance tuning. The kit brings power tuning to OS/2 users featuring a simple graphic user interface. It also tunes DOS/Windows applications running under OS/2, explaining all of OS/2's 55 DOS settings. The kit contains additional utilities to measure performance and tuning results, simplify bitmap viewing, automate file and desktop backup, monitor swap file growth with audible warning and simplified update, create a boot disc or partition, and more. It includes an additional bonus diskette packed with OS/2 bitmaps and a CPU Monitor Plus VL. The utilities are all written in REXX. Ten of the utilities have a GUI interface using VisProRexx from Hockware.

Clear & Simple Inc., P.O. Box 130,
W. Simsbury, CT 06092, 203-658-1204,
fax 203-651-0354.

QUERCUS SYSTEMS.

Personal REXX 3.0 for OS/2 is an

enhanced but fully compatible 32-bit implementation of the standard OS/2 REXX procedures language. It offers improved functionality, performance, support, and documentation. Its additional commands and REXXLIB functions perform array sorting, file system support, named pipes, interprocess communication, global variables, mathematics, text-mode user interfacing, and other operating-system services.

Personal REXX for Windows is a powerful, general-purpose language for quick programming tasks, application prototyping, and batch file programming. REXX has a programming interface that allows its use as a general-purpose embeddable macro language that enables any application to utilize REXX as its native scripting language.

REXXCOMM provides easy access to serial ports directly from REXX. Its high-level functions support sending and receiving data, matching multiple character strings, controlling a modem, and dialing numbers. Intermediate-level functions help manage port configuration parameters, time-out intervals, and RS232 control signals. File transfer support includes Xmodem, Ymodem, Kermit, and CompuServe-B.

REXXLIB is a function library that enhances the OS/2 REXX procedures language. It is an extensive toolkit, with full access to the advanced capabilities of OS/2, to speed development and prototyping of applications. Functions are provided for array manipulation and sorting, file system support, program execution, named pipes, interprocess communication, data conversion, mathematics, screen handling, and other system services.

Quercus Systems, P.O. Box 2157,
Saratoga, CA 95070, (408) 867-7399, fax
(408) 867-7489.

LEGENT CORP.

Prevail/XP Automation Point for OS/2 is a workstation-based automated operations product designed for external operation and management of host and distributed systems. Prevail/XP Automation Point is a general-purpose operator workstation that allows an operator to view and automatically respond to console messages from multiple systems at a single focal point. Automation is written using expert system-based rules and SAA REXX procedures. For unattended systems, Prevail/XP can notify remote technicians via alphanumeric pagers and

interactive voice calls.

Legent Corp., 2000 Park Lane,
Pittsburgh, PA 15275, (412) 494-1150,
fax (412) 494-1029.

SUITABLE ALTERNATIVES

REXX Diagnostic Commander's (RDC)

interactive full-screen source-level REXX debugger and diagnostic tool allows developers, administrative personnel, and all OS/2 professionals to code and test REXX Execs in a fully interactive environment. The interactive display allows the user to single step through any type of Exec written in REXX. The source code, variables, and logic flow can be altered anytime, usually with just a keystroke. Changes made to the source take immediate effect. Permanent changes are accommodated by the use of an external editor. RDC is compatible with IBM-supplied REXX for OS/2 Release 2.x or above. RDC also can be used on numerous other platforms, including MVS, VM/CMS, and DOS.

Suitable Alternatives, 4655 Old
Ironsides Dr., Suite 220, Santa Clara,
CA 95054, (408) 727-3142, fax (408) 727-
3170.

AMERICAN CODERS

RexxBASE provides a complete database for REXX programs. RexxBASE is an external DLL that is called using the REXX API. RexxBASE can read, write, update, DBase DBF, DBT, and NDX files. Other functions include SORT, JOIN, FILTERS, date handling, and more. RexxBASE can be used with OS/2 command files and with the popular REXX visual programming packages, VisPro/REXX, VX*REXX, and GpfREXX. It supports dBase III and IV DBF, DBT, NDX, and MDF files.

American Coders LTD., P.O. Box
97462, Raleigh, NC 27624, (919) 846-
2014.

STRATEGIC SOLUTIONS INTERNATIONAL CORP.

Service Point/32 provides REXX-based automation applications and a complete development environment. Create standalone or interactive scripts to command and control any vendor's element management system from NetView or SP/32. REXX language extensions replicate NetView's automation environment on OS/2 including the 3270-like VIEW panels, RUNCMD's, GENALERT's, and MSGs. RUNCMD scripts run in NetView or SP/32 without modification. Optional C language libraries





enable SP/32 to be embedded within other applications. It requires OS/2 and either Communications Manager/2 or ES Communications Manager but does not require NetView/PC.

Strategic Solutions International Corp., 1075 Tolland Turnpike, Manchester, CT 06040, (203) 649-1900, fax (203) 649-1230.

QIIQ LTD.

Terminal Manager adds multi-user capability to OS/2, allowing applications to run on low-cost terminals. Terminal Manager supports up to 48 terminals, each able to run up to 8 text-mode sessions. In each session, the user may run any required mix of 16-bit OS/2 programs, 32-bit OS/2 programs, and 32-bit REXX applications of most MSDOS programs. Operation of the host computer's screen and keyboard is unaffected. Terminals or PCs may be connected, either locally or remotely, to COM1-COM8 or to multiport adapters.

QIIQ Ltd., Elm House, 17-19 Claygate Lane, Surrey, KT7 ODL, 44 181 339 0739, fax 44 181 398 8443.

CIRRUS TECHNOLOGY INC.

Cirrus Technology's **Unite Image Viewer Object** provides the ability to view, manipulate, and compress/decompress color and gray-scale photographs and bi-tonal documents. Based on IBM's SOM technology, this object-oriented tool may be used via 32-bit APIs or with a front end GUI wrapper to integrate the object into various visual development environments, such as WATCOM's VX-REXX and IBM's Visual Age.

The **Unite Scanner Object**, which includes the Unite Image Viewer Object, is based on IBM's SOM technology and provides accessibility to functionality available in a wide range of scanners including Kodak, Fujitsu, Ricoh, and Hewlett-Packard. As a development tool for scanning applications, this object-oriented software may be used via 32-bit APIs or with a front end GUI wrapper to integrate the object into various visual development environments such as Watcom's VX-REXX and IBM's VisualAge. The Unite Scanner Object is developed for the IBM OS/2 environment and uses ANSI-standard Small Computer Systems Interface (SCSI) for connect-

ing peripheral devices and their controllers to microprocessors.

Cirrus Technology Inc., 5301 Buckeystown Pike, Frederick, MD 21701, 301-698-1900, fax 301-698-1909.

WATCOM

WATCOM's **VX*REXX** is an award-winning, easy to use, visual development environment for creating OS/2 applications with GUIs. VX*REXX combines a project management facility, visual designer, and an interactive source-level debugger to deliver a very approachable and highly productive visual development environment. Package your VX*REXX application as an EXE file or PM macro for royalty-free redistribution.

VX*REXX Client/Server Edition is a visual development environment for creating OS/2 graphical user interface applications. It includes all the features of VX*REXX V2.1 plus connection, query and chart objects that allow you to access databases, manipulate data and chart results quickly and easily. Features include drag-and-drop programming, bound controls, and professional multithreaded, multiwindowed application development. Package your VX*REXX application as an EXE file or PM macro for royalty-free redistribution.

WATCOM, 415 Phillip St., Waterloo, Ontario N2L 3X2, 519-886-3700, fax 519-747-4971.

HOCKWARE INC.

VisPro/REXX Bronze provides a low-priced entry point to OS/2 GUI programming. It is a full-featured, easy-to-use visual programming tool that lets you build 32-bit standalone GUI applications in record time. Features include: a visual programming environment, drag-and-drop programming that automatically generates REXX code, many CUA '91 programming objects, and a SOM toolkit for building custom objects.

VisPro/REXX Gold Edition takes the power of OS/2, Workplace Shell, and the REXX language and harnesses them into an easy-to-use visual programming environment. Build 32-bit standalone GUI applications in record time. It includes everything from the Bronze Edition, plus: a full-featured, animated graphical debugger, multiple CUA '91 views, access to a DB2

family of databases, the ability to print WYSIWYG, and tabular reports.

HockWare Inc., P.O. Box 336, Cary, NC 27512, (919) 380-0616, fax (919) 380-0757.

DIALOG SOFTWARE GMBH

WinMgr/PM provides an easy-to-use interface for developing standalone or client/server applications on local or wide area networks (using TCP/IP). It provides interfaces to the following languages: C, REXX/2, and Komet/PM, but the open design enables the use of any other programming language. WinMgr/PM can handle all the OS/2-PM user in-/output, so that your applications have only to react on a minimum set of events.

Dialog Software GmbH, Rosen-gartenplatz 7, 68161 Mannheim.

STRATEGIC SOLUTIONS INTERNATIONAL CORP.

MVS Application and SNA Network Management System for the NetImpact v. 3R1 OS/2-based PM/Workstation provides GUI for the MVS-based SNA/Agent that auto-discovers network and reports availability and reliability of managed elements. OS/2 REXX interface provides automation facilities for VTAM and MVS commands to address problems with SNA network and with MVS applications. Pricing is based on the SNA/Agent, which includes licenses for PM/Workstations with REXX automation facilities.

Strategic Solutions International Corp., 1075 Tolland Turnpike, Manchester, CT 06040, 203-649-1900, fax (203) 649-1230.

SIMMWARE,

REXXWARE is a robust enterprise management tool that can also be used to build a full suite of custom NetWare utilities. It facilitates your migration from NetWare 3.x to 4.1. REXXWARE gives you access to more than 240 NetWare operating-system functions. Features include: CLIB, Btrieve, Bindery, and Directory service calls for specific NetWare functions, a compiler, a Windows-based scheduler, and a dscripts that support access to remote servers.

Simmware, 2 Gurdwara Road, Ottawa, Ontario, Canada K2E 1A2, (613) 727-1779, fax (613) 727-3533.

A Special Report from the publishers of OS/2 Developer!

Quantities are limited.
Order now!
For fastest service,
fax your order to us
at 415-905-4967.



Smalltalk



☒ **YES!** Please send me _____ copies of Smalltalk!

Each copy is only \$9.95 in the US or \$12.95 in Canada.

Please add \$2.50 for shipping and handling.

- ☐ My check is enclosed
- ☐ Charge my credit card: ☐ Visa ☐ MasterCard ☐ American Express

Name _____

Signature _____

Date _____

Send them to:

Name _____

Address _____

City _____

State _____

Postal Code _____

Mail: Smalltalk
P. O. Box 7046
San Francisco
CA 94120

Phone: 1-800-444-4881

Fax: (415) 905-4967

Software Development '95 East

October 2-6, 1995 Washington, D.C.



Your Mind is Your Livelihood. Feed it at SD '95.

Software Development '95 is the most important event of the year for cutting-edge development professionals.

Interact with 10,000 of your peers and condense months of research and learning into a few short days. Over 150 lectures, workshops and tutorials featuring the industry's leading experts cover today's most relevant development topics:

- Team Managers and Management
- Managing for Quality and Productivity
- C++
- Windows 95/NT Development
- UI Design
- Building a Software Component Strategy
- Testing and Debugging



- Enterprise Application Re-Development
- Client/Server Architectures and Tools
- Database Programming and Design
- Object-Oriented Programming
- Building Open & Distributed Systems

Also see over 300 development tools from over 200 leading vendors and have your tough questions answered by the product experts.

Contact us today for the opportunity to increase your productivity, enhance the quality of your software, and feed your mind at SD '95.

Fax:
(415) 905-2222

Call:
(800) 441-8826

E-Mail:
sd95east@mfi.com

☐ Yes! Send me more information on Software Development '95 East.

Name

Title Company

Address

City State Zip

Phone Fax E-mail

Home Page <http://www.mfi.com/sdconfs/>

OS2D CODE

Reply today!

Enter your name
for a chance to
win a FREE
SD '95 VIP
Conference Pass.